Chapter 9: Main Memory

1

Chapter Topics

- Background
- Swapping
- Contiguous Memory Allocation
- Paging
- Structure of the Page Table
- Segmentation

2

Background

- A program must be brought (from disk) into memory for it to be executed.
  - Main memory, cache memory, and registers are only storage devices the CPU can access directly.
  - Registers are accessible in one CPU clock cycle.
  - Main memory access can take many CPU clock cycles.
  - Cache memory is used as a buffer between main memory and the CPU.
- Since main memory is shared by multiple processes, protection is required to ensure that each process will access only its legal memory addresses.

3

Address Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can happen at three different stages:
  - **Compile time**: If we know at compile time where the process will reside in main memory, then absolute code can be generated.
    - However, the code must be recompiled if the starting location in main memory changes.
  - **Load time**: If it is not known at compile time where the process will reside in main memory, then the compiler must generate relocatable code.
    - **Implies the contiguous memory allocation for each process.**
  - **Execution time**: If the process can be moved during its execution from one memory section to another, then binding must be delayed until run time.

4

Logical vs. Physical Address Space

- An address generated by the CPU is commonly referred to as a logical address, whereas an address seen by the memory unit is commonly referred as a physical address.
  - A logical address is also called a virtual address.
- A set of all logical addresses generated by a program is a logical address space.
- The set of all physical addresses corresponding to these logical addresses is a physical address space.

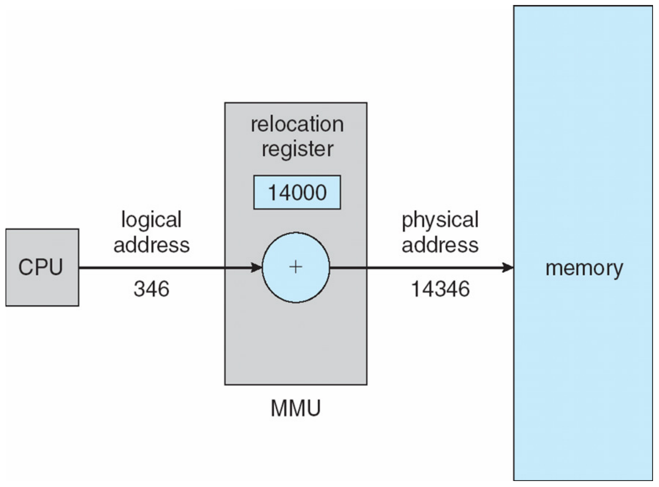
5

Memory-Management Unit (MMU)

- A memory-management unit (MMU) is a hardware device that maps logical addresses (also called virtual addresses) to physical addresses.
- If a process uses a contiguous memory space, a relocation register is used in the MMU, and the value in the relocation register is added to every logical address generated by a user process to map it to a physical address in main memory.

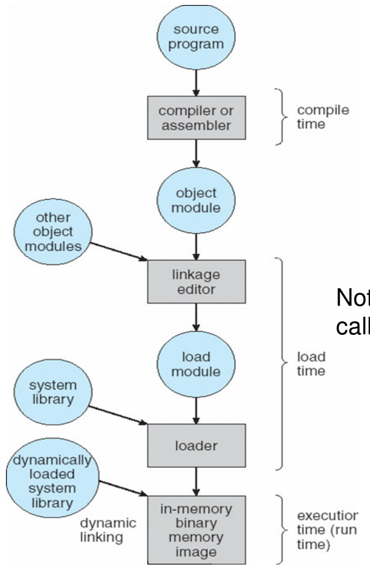
6

Dynamic Relocation Using a Relocation Register



7

Multistep Processing of a User Program



Note: *linkage editor* is also called *link editor* or *linker.*

8

Dynamic Loading

- With dynamic loading, a routine is not loaded into main memory until it is called.
  - The main program is loaded into memory and executed.
  - All routines are kept on disk in a relocatable format.
- We obtain better memory-space utilization because an unused routine is never loaded into memory.
- Useful when a large amount of codes are needed to handle the cases infrequently occurring, such as error routines.

9

Dynamic Linking and Shared Libraries

- If an OS supports only static linking, system libraries are treated like any other object modules and are combined by the loader (or linking loader) into the binary program image.
- In dynamic linking, linking is postponed until execution time.
- With dynamic linking, a *stub* is included in the program image for each library routine reference.
  - The stub is a small piece of code; and when it is executed, it checks whether the needed routine is already in memory. If it is not in memory, it is loaded into memory.
  - The stub replaces itself with the address of the routine, and executes the routine.
- Dynamic linking is particularly useful for shared libraries, such as standard C language libraries, because a single copy of each routine loaded into main memory can be shared by different user programs.
  - Dynamic Link Libraries (DLLs) are used in Windows and Linux systems.

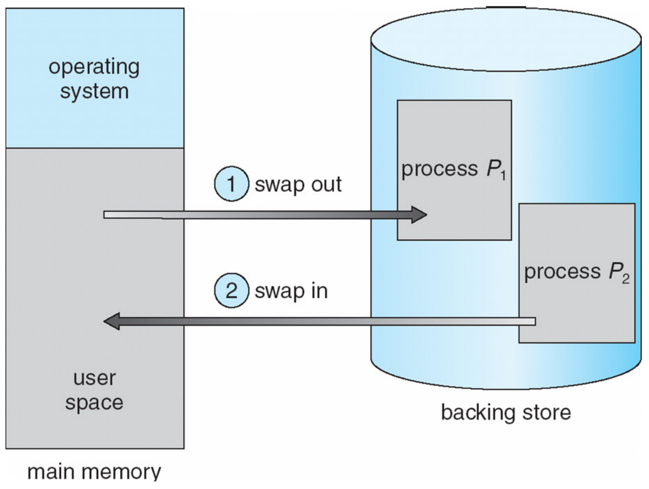
10

Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.
- The backing store is usually a partition (called a swap space) of a disk large enough to accommodate the copies of memory images for all users, and must provide direct access to these memory images.
- Examples of swapping policy:
  - If a round-robin CPU scheduling algorithm is used, when each process finishes its CPU time slice (aka CPU time quantum), it could be swapped with another process.
  - If a priority-based CPU scheduling algorithm is used, when a higher-priority process arrives, a lower-priority process could be swapped out so the higher-priority process can be loaded and executed. This swapping scheme is sometimes called **roll out, roll in.**
- The major part of the swap time is transfer time between main memory and the backing store, which is directly proportional to the amount of memory swapped.

11

Swapping of Two Processes



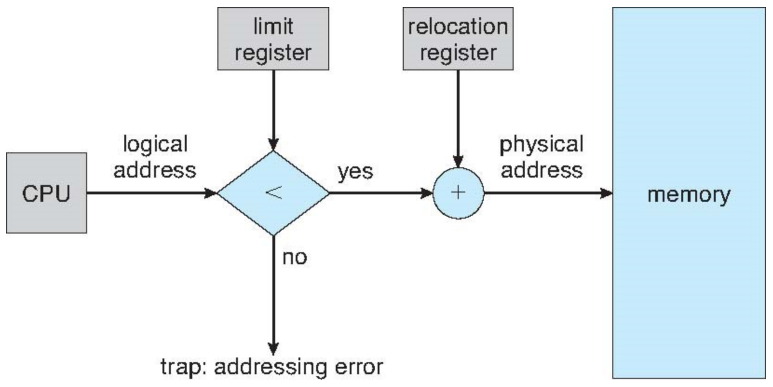
12

Contiguous Memory Allocation

- In contiguous memory allocation, each process is contained in a single contiguous section of memory.
- Relocation and limit registers are used to protect the user processes from each other.
  - The relocation register contains the value of smallest physical (i.e., main memory) address of the process currently running.
  - The limit register contains the range (i.e., total number) of logical addresses of the process — each logical address must be less than the value of the limit register.

13

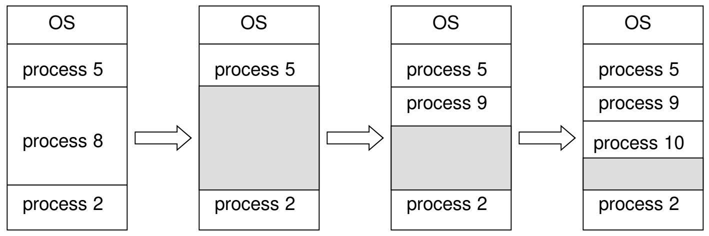
Hardware Support for Relocation and Limit Registers



14

Contiguous Memory Allocation (cont'd)

- Variable-partition scheme
  - A hole is a contiguous section of memory that is available. So, holes of various sizes could be scattered throughout memory.
  - When a process arrives, it is allocated a hole large enough to accommodate it.
  - The OS maintains a table containing information about allocated partitions and free partitions (i.e., holes)



15

Dynamic Storage-Allocation Problem

- How to satisfy a request of certain size from a list of holes?
  - **First-fit**: Allocate the *first* hole that is big enough.
  - **Best-fit**: Allocate the *smallest* hole that is big enough.
    - We must search the entire list of holes, unless the holes are ordered by size in the list.
    - This strategy produces the smallest leftover hole.
  - **Worst-fit**: Allocate the *largest* hole.
    - This strategy produces the largest leftover hole, such that it can easily accommodate another process.
- Both first-fit and best-fit strategies are better than worst-fit strategy in terms of speed or storage utilization.

16

Fragmentation

- **External Fragmentation:** The free memory spaces exist between allocated memory partitions, such that no contiguous free memory space is larger than the process size requested.
- **Internal Fragmentation:** A fixed-size memory partition allocated to a process may be slightly larger than the process size. This size difference results in unused memory that is internal to a fixed partition.
- External fragmentation problem can be reduced by compaction:
  - Compaction is possible only if relocation of processes is dynamic and can be done at execution time.

17

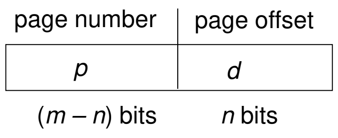
Paging

- Paging is a memory management scheme that permits the physical address space of a process can be noncontiguous:
  - Dividing the physical memory into fixed-sized blocks called **page frames** (or simply **frames**).
  - Dividing the logical address space into blocks of same size called **virtual pages** (or simply **pages**). The size of a page is power of 2, typically between 512 bytes and 8 KB.
  - If the virtual page containing the referenced word is not in main memory, it should be fetched from a disk and loaded into a page frame.
- A page table is used to translate the logical addresses to physical addresses (i.e., from virtual page numbers to page frame numbers).
- When paging is used, there is no external fragmentation problem, but internal fragmentation within a page frame (containing the last page of a process) is possible.

18

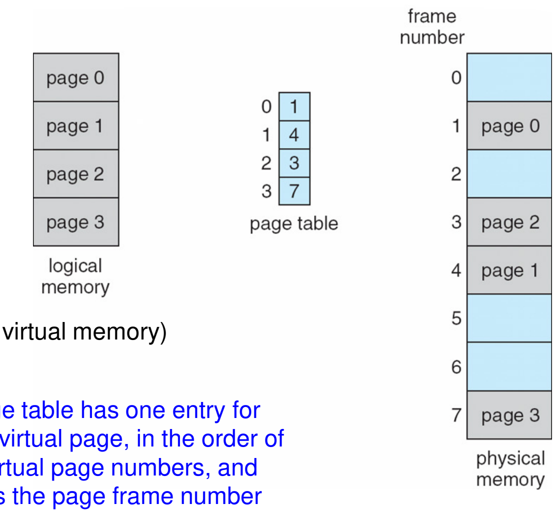
Address Translation Scheme in Paging

- The logical address generated by the CPU is divided into two parts:
  - **Page number (*p*):** used as an index into a *page table* which contains the page frame number (or the disk block number if the page is not in main memory) of each virtual page.
  - **Page offset (or displacement) (*d*):** the word number (or byte number) within the virtual page.
  - For a given logical address space of 2^m and a page size of 2^n:



19

Paging Model of Logical and Physical Address Spaces

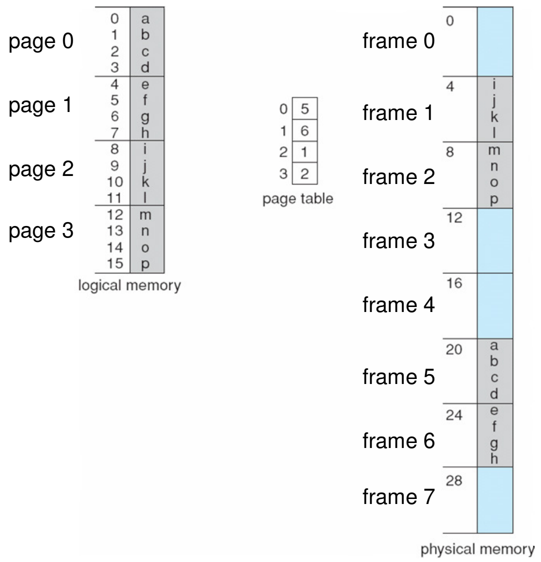


(or virtual memory)

A page table has one entry for each virtual page, in the order of the virtual page numbers, and stores the page frame number where the virtual page is stored.

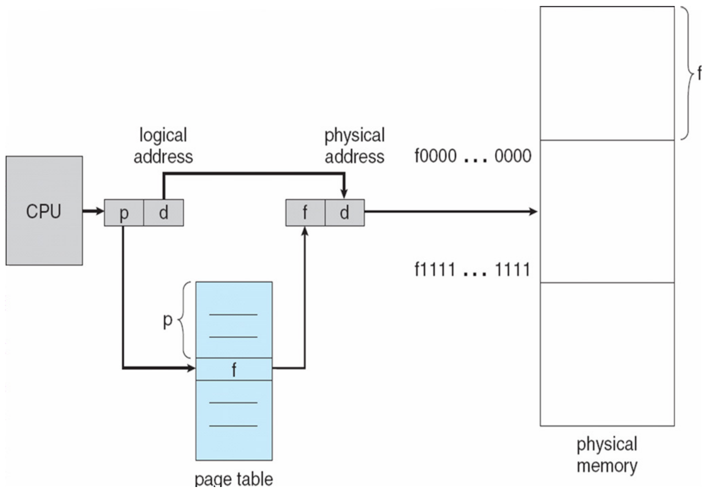
20

Paging Example



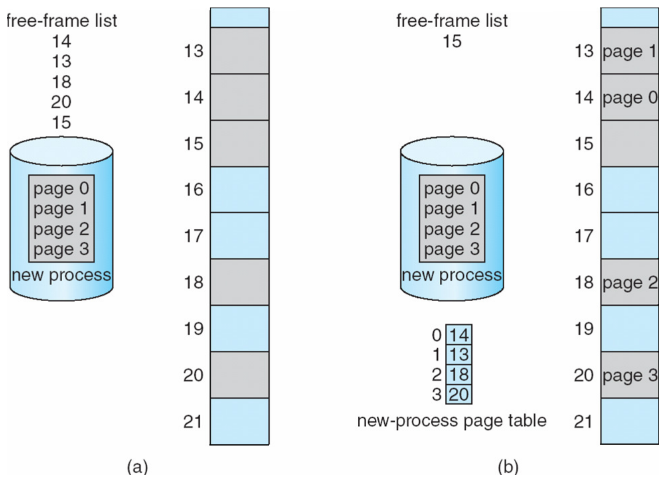
- 32-byte main memory and 4-byte pages just for illustration purpose.

Address Translation



f is the frame number in binary, and the starting address of the frame number f is f0000 …. 0000 (concatenated with n 0's when the frame size is 2^n).

Free Page Frames in Main Memory

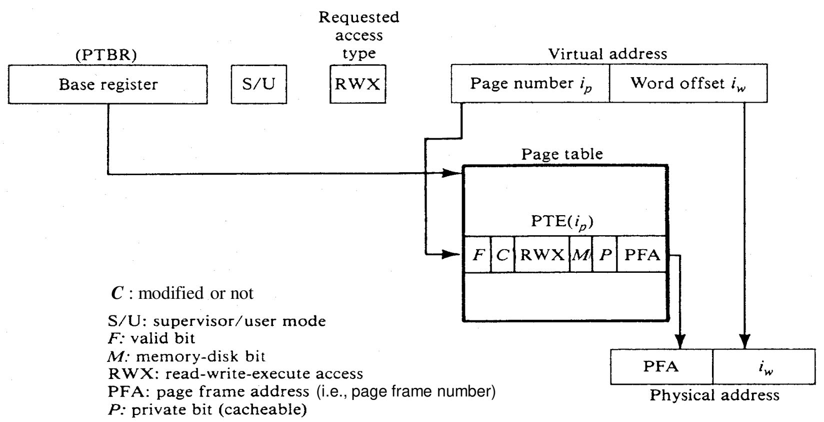


Before allocation After allocation

Implementation of a Page Table

- Each page table is kept in main memory.
- **Page Table Base Register (PTBR)** points to the base address (i.e., starting location) of a page table in main memory.
- **Page Table Length Register (PTLR)** indicates the size of a page table (i.e., the total number of entries in the page table).
- In this scheme, accessing a data/instruction word requires two memory accesses — one for the page table entry and one for the data/instruction word.

Implementation of a Page Table



C : modified or not

S/U: supervisor/user mode
F: valid bit
M: memory-disk bit
RWX: read-write-execute access
PFA: page frame address (i.e., page frame number)
P: private bit (cacheable)

If PTE(i_p) • $F = 1/0 \rightarrow$ page exist/null page
If PTE(i_p) • $RWX =$ RWX $\rightarrow$ valid access
If PTE(i_p) • $M = 1 \rightarrow$ PTE(i_p)•PFA is page address in main memory
else it is disk address of page
If PTE (i_p) • $P = 1 \rightarrow$ page private

Virtual to real page address translation.

25

Translation Look-aside Buffer (TLB)

- The problem of accessing memory twice (when we use the page table) can be partially solved by using of a special fast-lookup hardware called Translation Look-aside Buffer (TLB) in the processor.
  - A TLB stores a subset of the entries of the page table of currently running process.

Page #	Frame #
20	100
10	250
13	105
5	120

- Some TLB also stores the process ID (if the TLB is shared by multiple processes) and the allowed access mode (i.e., RWX) in each entry.
- Associative memory (aka content addressable memory) allows to locate a matching word without generating its address, by searching the words stored in the associative memory in parallel. So, the TLB can be implemented using an associative memory.

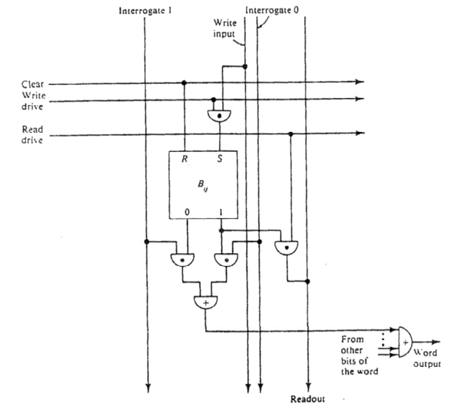
26

A Typical Cell in an Associative Memory

Interrogation information	Mask*	Information stored 0	Information stored 1
0	0	0	0
1	0	0	0
0	1	0	1
1	1	1	0

"0" output if matches

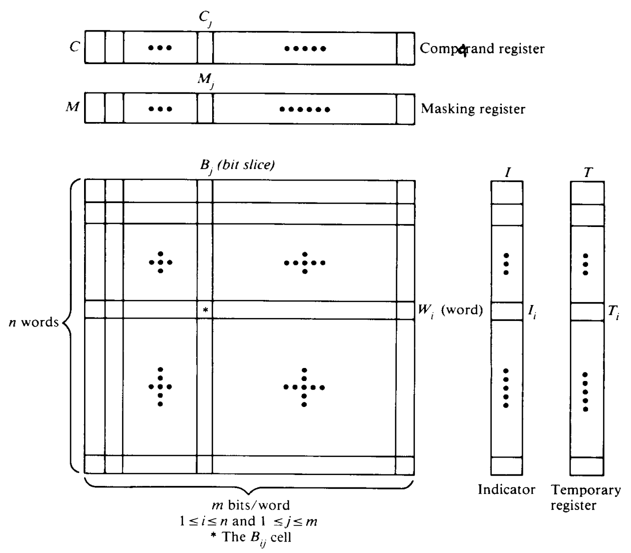
*Mask = 0 means that no comparison is performed at that bit position for all words.



The schematic logic design of a typical cell in an associative memory.

27

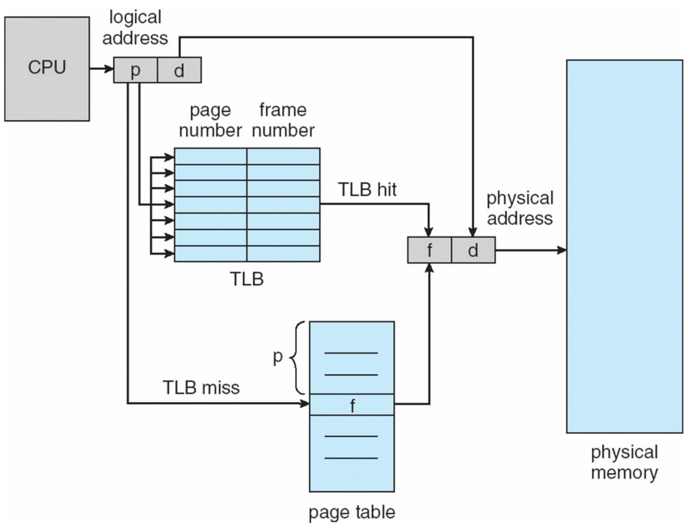
Associative Memory Structure



An associative memory array and working registers.

28

Paging Hardware with TLB



29

Effective Access Time

- Associative TLB Lookup time = Tc
- Memory cycle time = Tm
- TLB hit ratio: probability that a (virtual) page number (containing the referenced word) is found in the TLB, so that we can identify the memory frame number where the page is stored.
- If the TLB hit ratio is denoted by α, Effective Access Time (EAT) of a referenced instruction/data word:

$$EAT = \alpha(Tc + Tm) + (1 - \alpha)(Tc + 2Tm)$$
$$= Tc + \alpha Tm + (1 - \alpha)2Tm$$

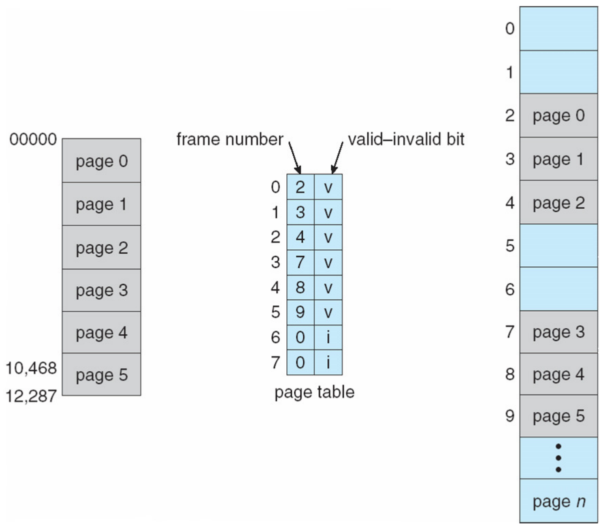
30

Memory Protection

- **Valid-invalid** bit is included in each entry in the page table:
  - "valid" indicates that the page is within the process' logical address space, and is thus a legal page.
  - "invalid" indicates that the page is not in the process' logical address space.
- There are other cases, the valid-invalid bit of a virtual page is marked as invalid:
  - For example, a copy of a shared page (separately stored in another local memory of a processor) is modified by another process.

31

Valid (v) or Invalid (i) Bit in a Page Table



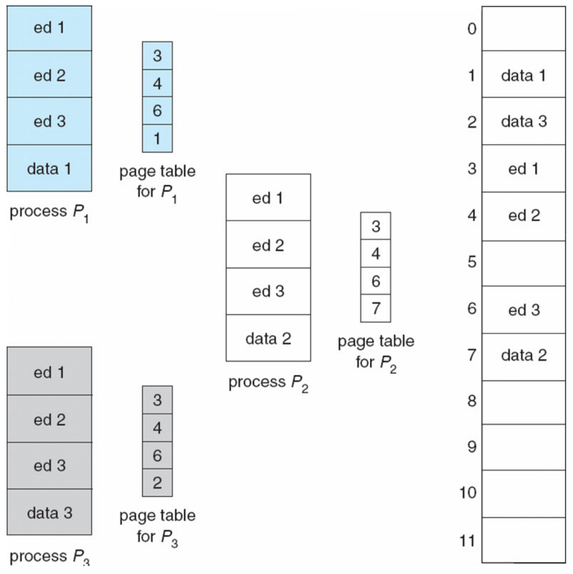
32

Shared Pages

- Shared code
  - One copy of reentrant (i.e., non-self-modifying) code can be shared by processes.
  - For example, text editors, compilers, window systems, run-time libraries, etc.
- Private data
  - Each process keeps a separate copy of its private data.
- Sharing pages between processes is difficult due to aliasing problem, as each process use its own sequential virtual pages numbers 0, 1, 2, 3, …
  - For example page 10 of process 1 and page 10 of process 2 may not be the same. On the other hand, page 10 of process 1 may be the same as page 20 of process 2.

33

An Example of Shared Pages



34

Structure of the Page Table

- Hierarchical Page Tables (aka Multilevel Page Tables)
- Hashed Page Table
- Inverted Page Table

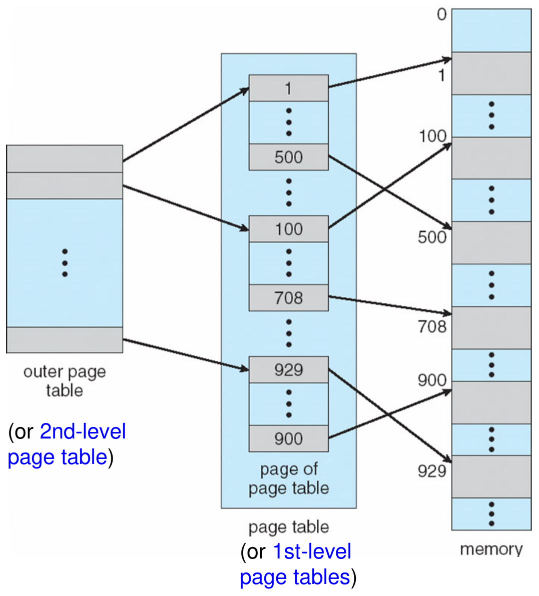
35

Hierarchical Page Tables

- Two-level page table: Divide the original page table into subtables of the same size, which can be called 1st-level page tables. And the 2nd-level page table (aka *outer page table*) is created such that each entry in the 2nd-level page table points to a 1st-level page table.
  - Only the 2nd-level page table is resident in the main memory, whereas a subset of 1st-level page tables are resident in the main memory.
- Multilevel page table: The single 2nd-level page table is divided into multiple 2nd-level page tables, and the 3rd-level page table is created to reference the 2nd-level page tables, and so on.

36

Two-Level Page Table Scheme



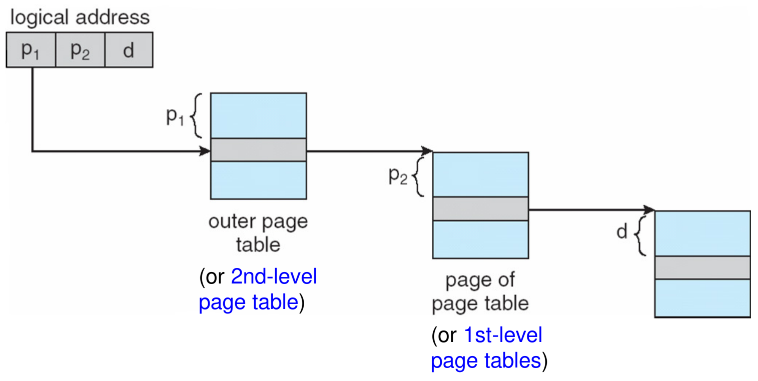
Two-Level Paging Example

- A 32-bit logical address is divided into:
  - a page number consisting of high-order 22 bits.
  - a page offset consisting of low-order 10 bits.
- If the page table is divided into 1st-level page tables, each of which contains 1024 entries, the 22-bit page number is divided into:
  - a 12-bit entry address in the 2nd-level page table, and it corresponds to a 1st-level page table number.
  - a 10-bit entry address in the selected 1st-level page table.
- Then, a logical address is as follows:

page number		page offset
p_1	p_2	d
12	10	10

where p_1 is the entry number (to look up) in the 2nd-level page table, and p_2 is the entry number (to look up) in the selected 1st-level page table.

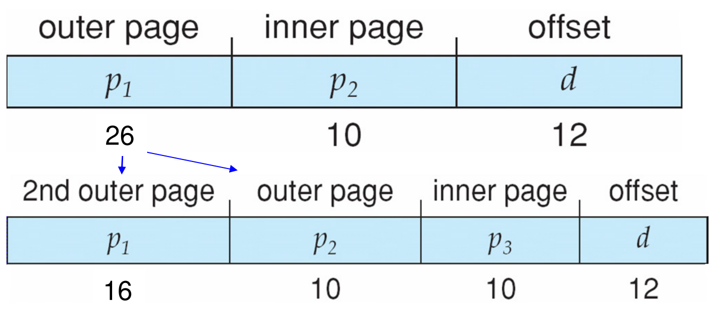
Address-Translation Scheme



39

Three-level Paging Scheme

- The outer page table (i.e., 2nd-level page table) in the two-level paging is divided, and the 3rd-level page table is added.



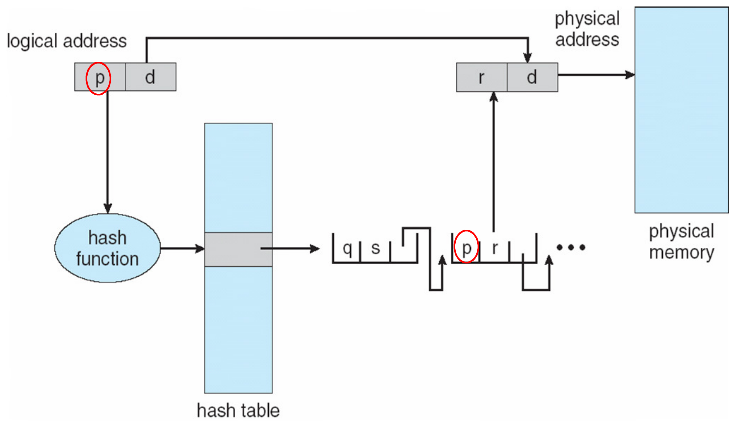
40

Hashed Page Tables

- A hashed page table is commonly used for handling a large logical address space (e.g., logical address larger than 32 bits) because multilevel paging requires multiple memory accesses, one for each level.
- The virtual page number is hashed into an entry of the hashed page table.
  - This entry contains a linked list of elements, where each element consists of three fields:
    - (1) the virtual page number (hashed to that entry),
    - (2) page frame number (that is allocated to store the virtual page), and
    - (3) a pointer to the next element in the linked list.
  - So, each entry of the hashed page table contains a group of (virtual page number, page frame number) pairs, where all those virtual page numbers have the same hash value.

41

Hashed Page Table (cont'd)



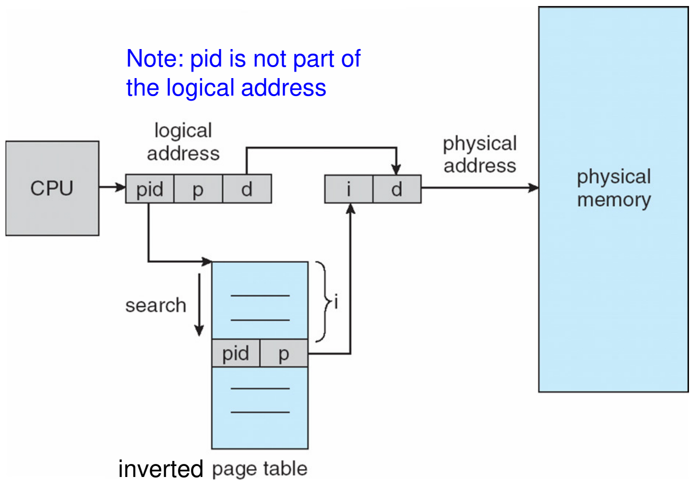
42

Inverted Page Table (IPT)

- The inverted page table has one entry for each page frame in the main memory, and the entry contains (1) the virtual page number that is stored in the corresponding page frame (if exists), (2) ID of the process that owns the virtual page, (3) allowed access mode (i.e., RWX), modified bit (aka dirty bit), etc.
- So, the size of inverted page table is fixed, and every process (sharing the memory) can use it (instead of using its own page table).
- In the inverted page table, since the entries are in the order of page frame numbers, searching of the entries is required with the pid of the current process and the virtual page number containing the referenced word, in order to find the entry containing the referenced virtual page number.
  – A hash table can be used to limit the search to one or at most a few entries in IPT: a virtual page number (and the pid) is hashed to an entry in the hash table, and each entry contains the IDs of the frames that store the virtual pages whose numbers are hashed to that entry.

43

Inverted Page Table (IPT) Architecture



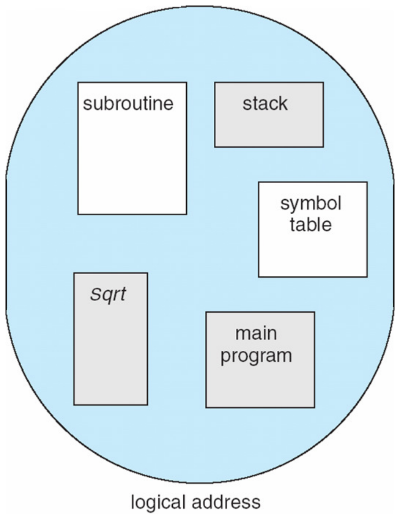
44

Segment

- A program is a collection of segments, where each segment is a set of logically related contiguous elements in a logical address space, such as:
  - main program
  - procedure
  - function
  - method
  - object
  - local variables, global variables
  - common block
  - stack
  - symbol table
  - array

45

User's View of a Program



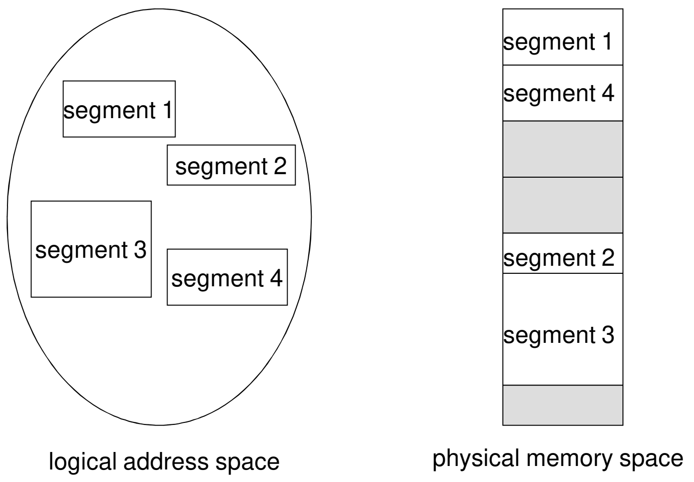
46

Segmentation

- A segment can be easily shared by multiple processes as the segments are referenced by their unique names in the programs, and each segment name is converted to a segment number (to generate logical addresses) at the compile time or loading time.
- In the segmented memory system, each segment is stored contiguously in the main memory.
  - So, the whole segment needs to be fetched from disk to main memory even though only a small fraction of it would be referenced by the process during the CPU time quantum.

47

Segmentation (cont'd)

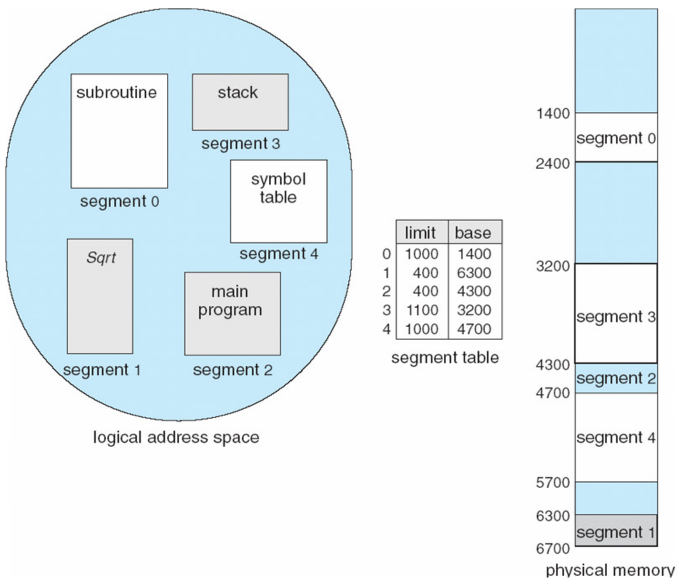


48

Segmentation Architecture

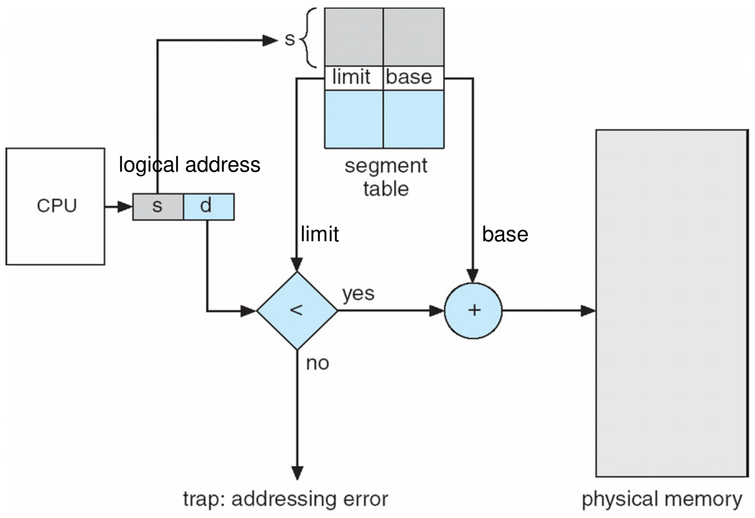
- A logical address consists of two parts:
  <segment number, offset (aka displacement) in the segment>
- Each process has a **segment table** which has one entry for each segment, and each entry contains:
  - **Base (address):** the starting physical address where the segment resides.
  - **Limit:** the length of the segment (in terms of number of bytes or words that is addressable).
  - Allowed access mode (i.e., RWX).
  - One bit indicating whether the segment is currently in memory or not.
  - One bit (called modified bit or dirty bit) indicating whether the content of the data segment has been modified or not while it has been in the memory.
- Segment Table Base Register (STBR) points to the segment table's location (i.e., base address) in memory.

Example of Segmentation



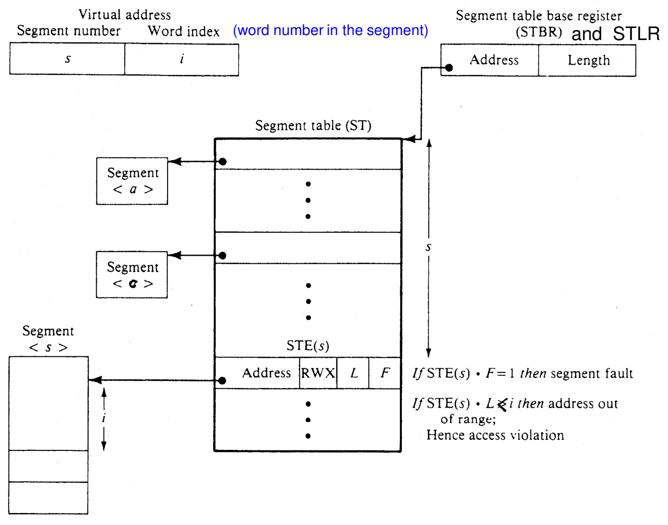
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Segmentation Hardware



51

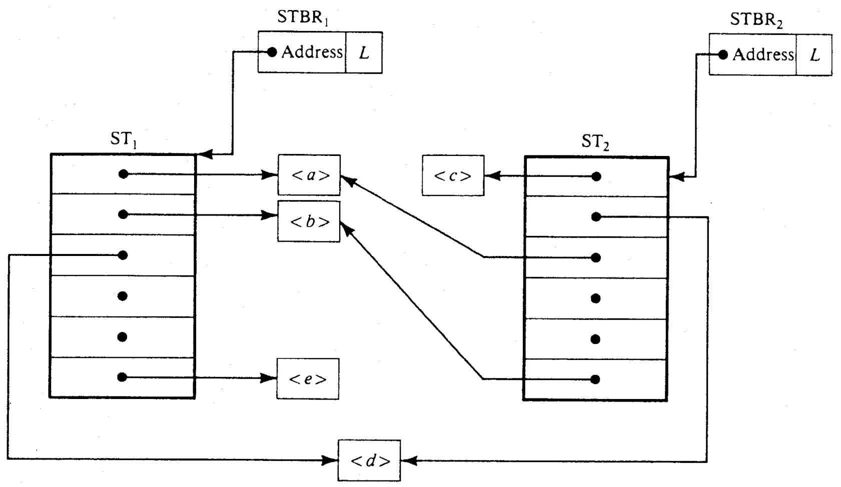
Address Mapping in Segmented Memory System



Address mapping in a segmented system.

52

Sharing of Segments between Processes



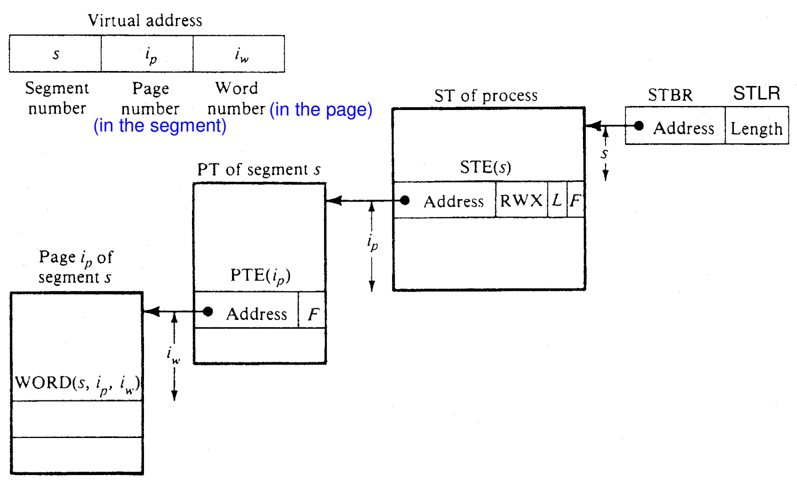
Sharing of segments by two active processes.

Paged Segments

- Each segment can be partitioned into pages, so that the individual pages can be fetched into memory as they are referenced one by one, instead of fetching the whole segment.
- The pages of a segment don't need to be stored consecutively in memory as they are managed by a page table of the segment.
- Each segment has a page table (if the segment size is larger than the page size).
  - The page table of a segment has entries for the virtual pages within the segment.
- Same virtual address is used in both segmentation and paged segmentation; only the interpretation would be different --- <segment #, word # in the segment> or <segment #, page# in the segment, word # in the page>.

54

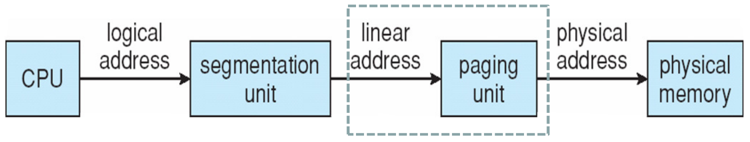
Address Mapping for Paged Segments



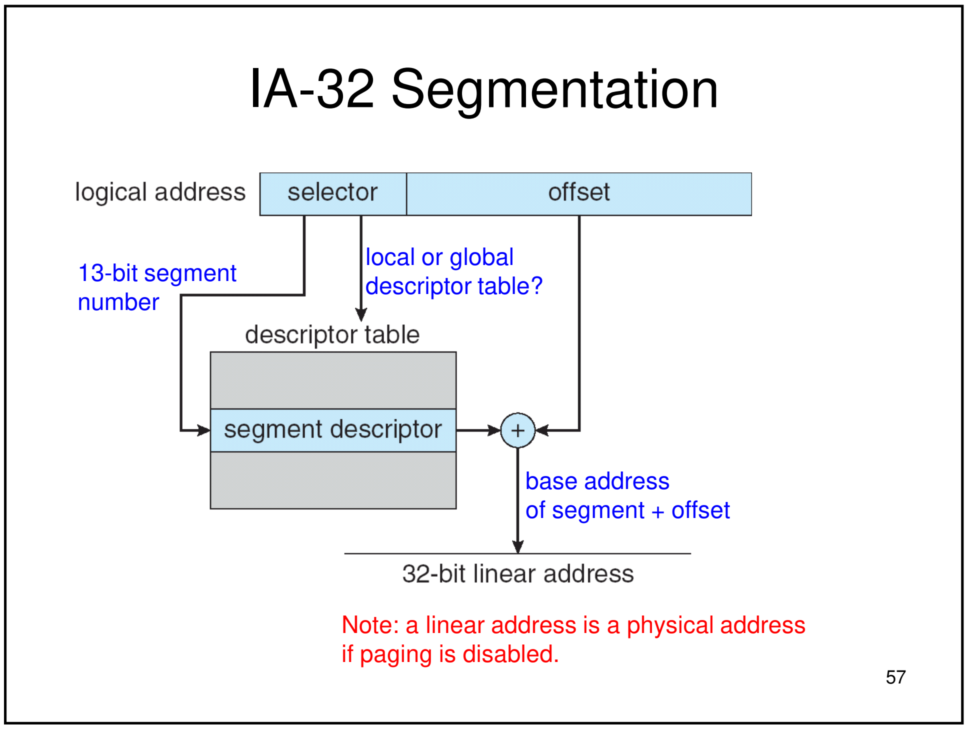
Address mapping in a system with paged segments.

Example: IA-32 Architecture (e.g., Intel Pentium)

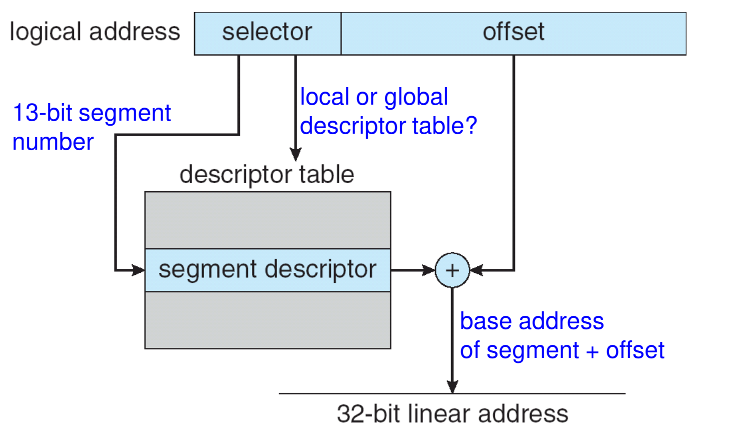
- Supports both (pure) segmentation and segmentation with paging (i.e., paged segments).
- CPU generates a logical address:
  - The logical address is given to the segmentation unit which produces a linear address.
  - A linear address is a physical address if paging is disabled. Otherwise, it contains the page number within the segment and the offset in the page.
  - If the paging unit is enabled, it generates a physical address in main memory.



56

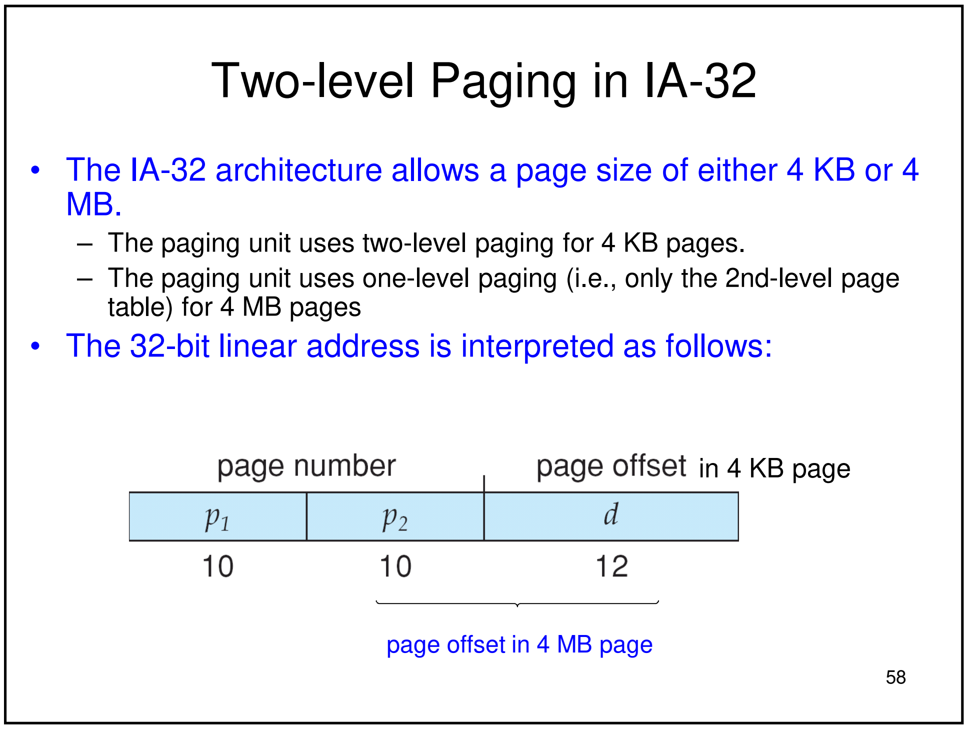


IA-32 Segmentation



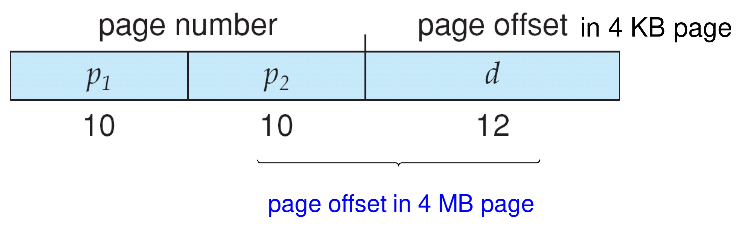
Note: a linear address is a physical address if paging is disabled.

57



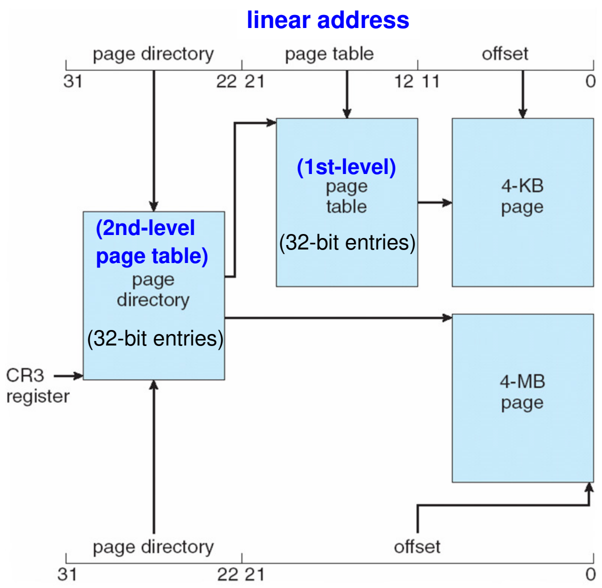
Two-level Paging in IA-32

- The IA-32 architecture allows a page size of either 4 KB or 4 MB.
  - The paging unit uses two-level paging for 4 KB pages.
  - The paging unit uses one-level paging (i.e., only the 2nd-level page table) for 4 MB pages
- The 32-bit linear address is interpreted as follows:



58

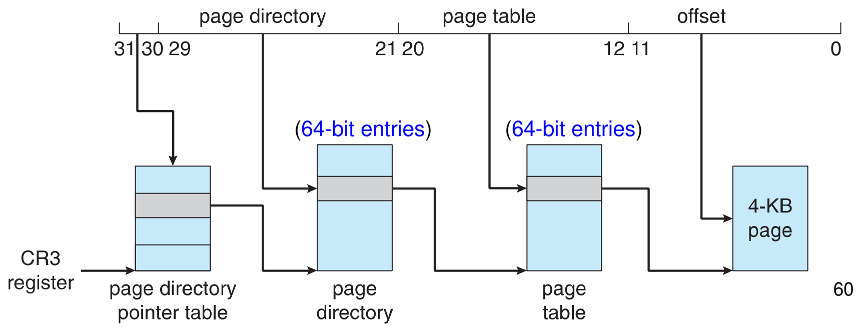
Paging in the IA-32 Architecture



59

Page Address Extension (PAE)

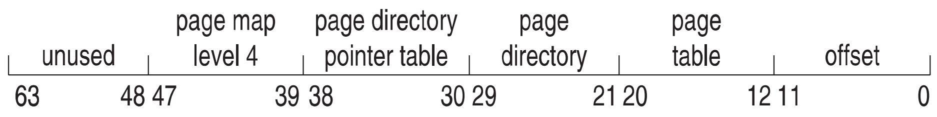
- Page Address Extension (PAE) is a set of h/w specifications, and when it is enabled, 32-bit processors can access a physical address space larger than 4 GB by increasing the size of page-directory entries and page-table entries from 32 bits to 64 bits, which allows the base address of a page table and a page frame to extend from 20 to 24 bits.
  - Adding PAE support to IA-32 increased the memory address from 32 bits to 36 bits, which supports up to 64 GB of memory space, by mapping 32-bit virtual address to 36-bit memory address.
  - An application can swap in and out different parts of memory into the visible address space to make use of more than 4 GB of RAM, but it can only see (i.e., access) maximum 4GB at any single point in time.
- PAE uses 3-level paging scheme, and supports 4 KB pages and 2MB pages.



60

Paging in x86-64 (aka x64 and AMD64) Architecture

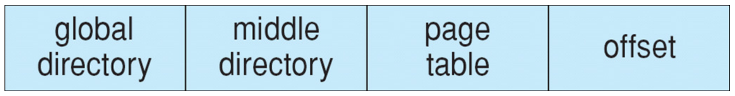
- x86-64 is a 64-bit architecture, but currently it provides a 48-bit virtual address with support for page sizes of 4 KB, 2 MB, and 1 GB using 4 levels of paging hierarchy.



61

Three-level Paging in Linux

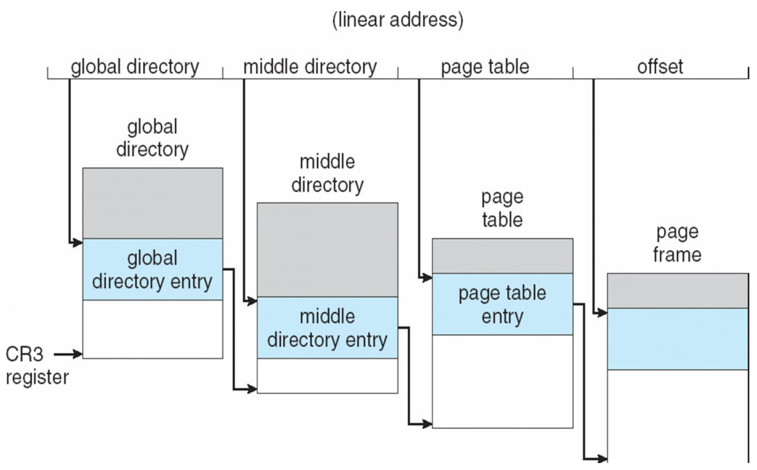
- A logical address is broken into four parts:



62

Three-level Paging of Linux on Pentium

- Since Pentium architecture uses two-level paging, the middle directory entry number is in 0 bits, bypassing the middle directory.



63