# Paging and Segmentation

CS 256/456
Dept. of Computer Science, University of Rochester

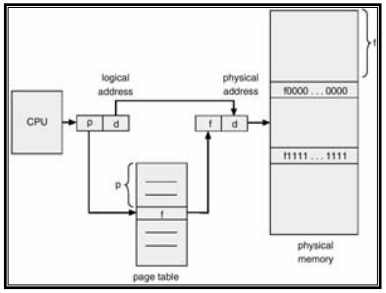


## Paging: Address Translation Scheme

A logical address is divided into:

- *Page number (p)* – used as an index into a *page table* which contains base address of each page in physical memory.
- *Page offset (d)* – the offset address with each page/frame.

Use TLB to speed up the address translation.






## Memory Access Setting in Page Table

- Parts of the logical address space may not be mapped
  - *Valid-invalid* bit attached to each entry in the page table.
  - indicating whether the associated page is in the process' logical address space, and is thus a legal page.
- Some pages are read-only, or can't contain executable code
  - access bits in page table to reflect these.
- Software exception if attempting to access an invalid page, or to perform disallowed actions



## Process Creation: Copy-on-Write

- Basic idea:
  - fork() semantics says the child process has duplicate copy of the parent's address space
  - child process often calls exec() right after fork()
  - Copy-on-Write (COW) allows both parent and child processes to initially *share* the same pages in memory.
- Implementation:
  - shared pages are marked readonly after fork().
  - if either process modifies a shared page, a page fault occurs and then the page is copied.
  - the other process (who later faults on write) discovers it is the only owner; so it doesn't copy again.

## Page Table Structure

- Problem with a flat linear page table
  - assume a page table entry is 4-byte; page size is 4KB; the 32-bit address space is 4GB large
  - how big is the flat linear page table?
- Solutions:
  - Hierarchical Page Tables
    - break the logical page number into multiple levels
- Metrics:
  - Space consumption and lookup speed

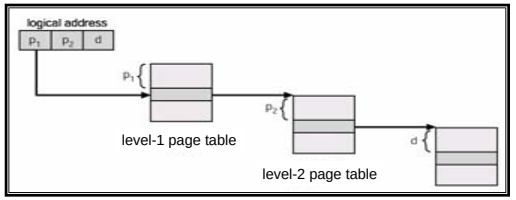


## Two-Level Page Table

- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page offset consisting of 12 bits.
  - a page number consisting of 20 bits; further divided into:
    - a 10-bit level-2 page number.
    - a 10-bit level-1 page number.
- Thus, a logical address look likes:

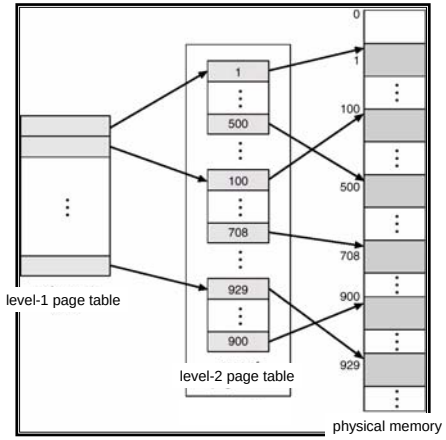
| page number | | page offset |
|---|---|---|
| $p_i$ | $p_2$ | $d$ |
| 10 | 10 | 12 |

- Address translation scheme:






## Two-Level Page Table: An Example

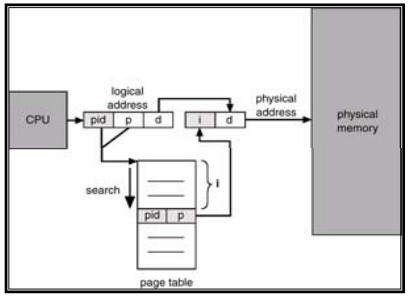



- Space consumption
- Lookup speed



## Deal With 64-bit Address Space

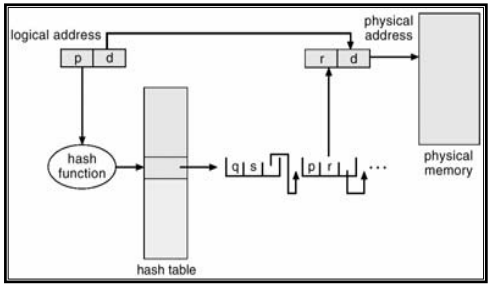
- Two-level page tables for 64-bit address space
  - more levels are needed
- Inverted page tables
  - One entry for each real page of memory.
  - Entry consists of the process id and virtual address of the page stored in that real memory location.




- Problems:
  - search takes too long
  - difficult to share memory

## Hashed Page Tables

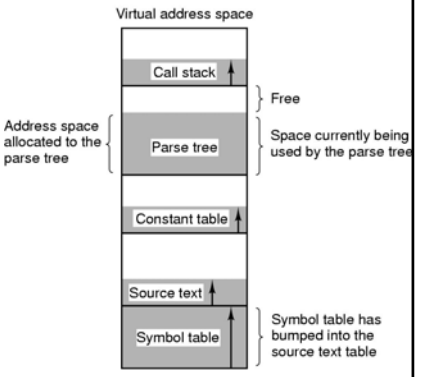
- The virtual page number is hashed into a page table. This page table contains a chain of elements hashing to the same location.
- Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.





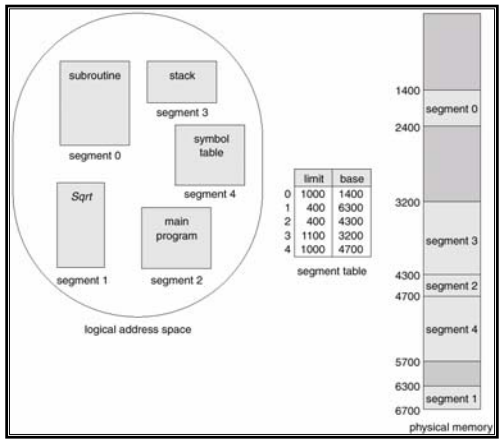

## Segmentation

- One-dimensional address space with growing pieces
- At compile time, one table may bump into another
- Segmentation:
  - generate segmented logical address at compile time
  - segmented logical address is translated into physical address at execution time





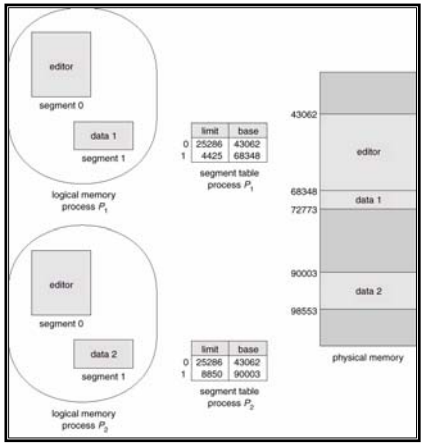

## Example of Segmentation





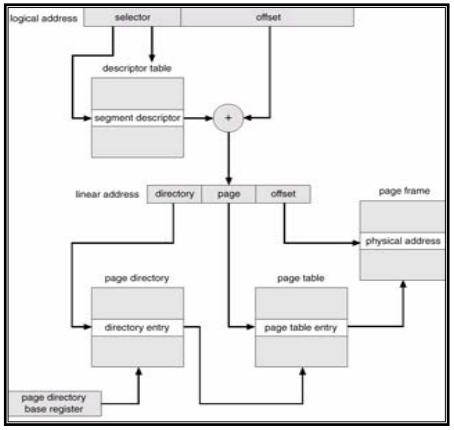

## Sharing of Segments

- Convenient sharing of libraries



## Segmentation & Paging – Intel x86

- Segmentation and paging with a two-level paging scheme.






## Virtual Memory

- **Virtual memory** – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Allows address spaces to be shared by several processes.
  - Copy-on-write: allows for more efficient process creation.
- Demand paging
  - Make a physical instance of a page in memory only when needed.

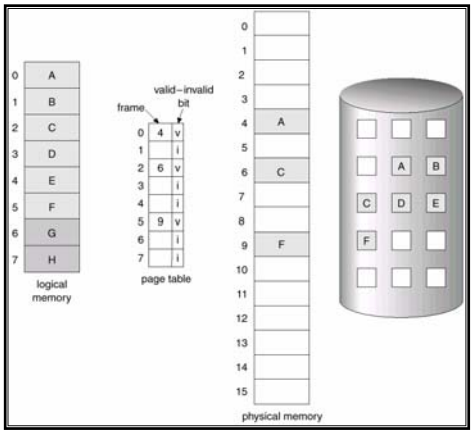


## Backing Store

- With virtual memory, the whole address space of each process has a copy in the backing store (i.e., disk)
  - program code
  - data/stack
- Consider the whole program actually resides on the backing store, only part of it is cached in memory.
- With each page table entry a valid-invalid bit is associated (1 ⇒ in-memory, 0 ⇒ not-in-memory or invalid logical page)



## Page Table When Some Pages Are Not in Main Memory



## Disclaimer

- Parts of the lecture slides contain original work of Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Andrew S. Tanenbaum, and Gary Nutt. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).