Betriebssysteme

Memory Management

Lehrstuhl Systemarchitektur

WS 2009/2010

Memory Management Strategies

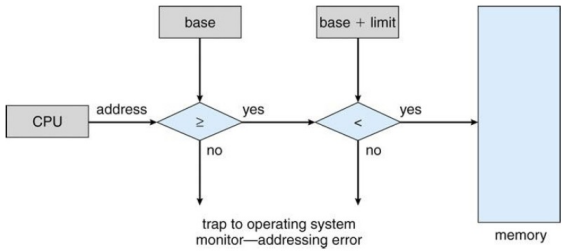
- Background
- Swapping
- Allocation
- Relocation
- Segmentation
- Paging

Background

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are the only storage that the CPU can access directly
- Register access in one CPU clock (or less)
- Main memory can take many cycles
- Cache sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

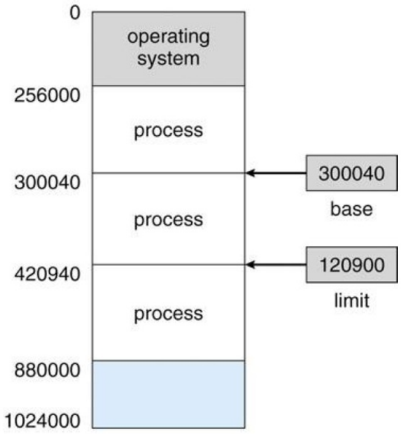
Memory Partitioning

- Main memory usually split into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes held in high memory
- Registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses



Simple Protection with Base and Limit Registers

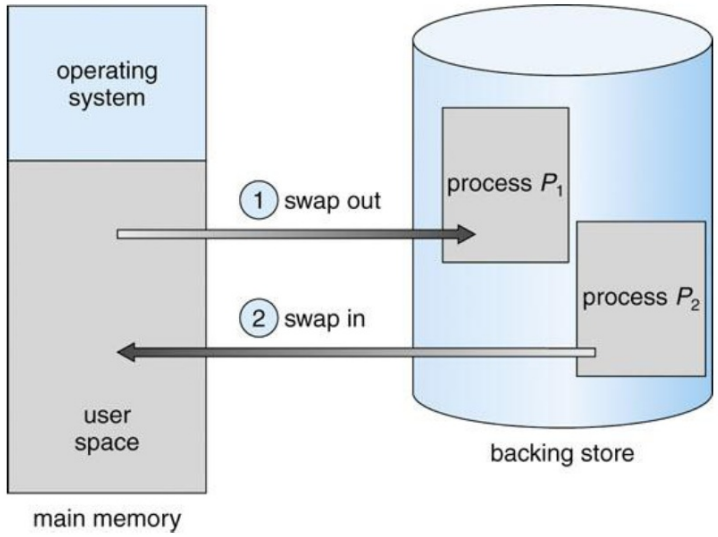
- A pair of base and limit registers define the logical address space



Swapping

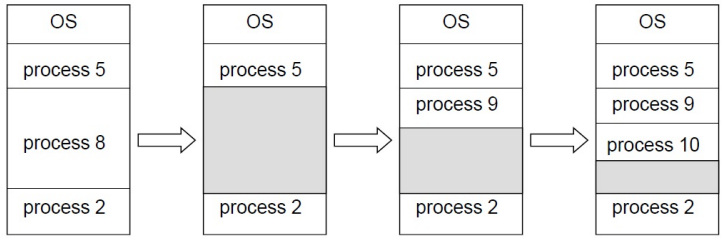
- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
- Backing store - fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- Roll out, roll in - swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a ready queue of ready-to-run processes which have memory images on disk
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

Schematic View of Swapping



Contiguous Allocation

- Multiple-partition allocation
  - Hole - block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Operating system maintains information about:
    - allocated partitions
    - free partitions (hole)



Dynamic Storage-Allocation Problem

How to satisfy a request of size n from a list of free holes

- First-fit: Allocate the first hole that is big enough; fastest allocation policy, produces leftover holes of variable size
- Best-fit: Allocate the smallest hole that is big enough; must search entire list, unless ordered by size
- Worst-fit: Allocate the largest hole; must also search entire list; Produces the largest leftover hole

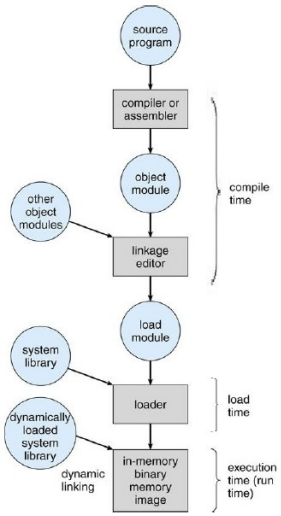
Fragmentation

- External Fragmentation - total memory space exists to satisfy a request, but it is not contiguous
- Internal Fragmentation - allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible only if relocation is dynamic, and is done at execution time
  - I/O problem
    - Lock job in memory while it is involved in I/O
    - Do I/O only into OS buffers

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 memory location known a priori, absolute code can be generated;
    ➜ must recompile code if starting location changes
  - Load time: Must generate relocatable code, if memory location is not known at compile time
  - Execution time: Binding delayed until run time, if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers)

Multistep Processing of a User Program



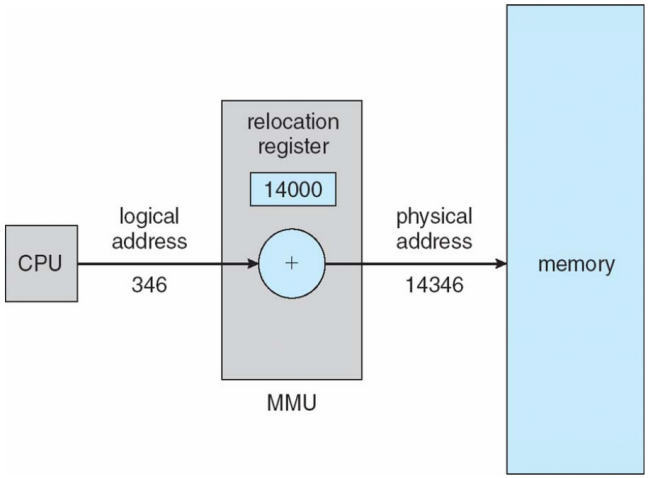
Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management
  - Logical address - generated by the CPU; also referred to as virtual address
  - Physical address - address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme

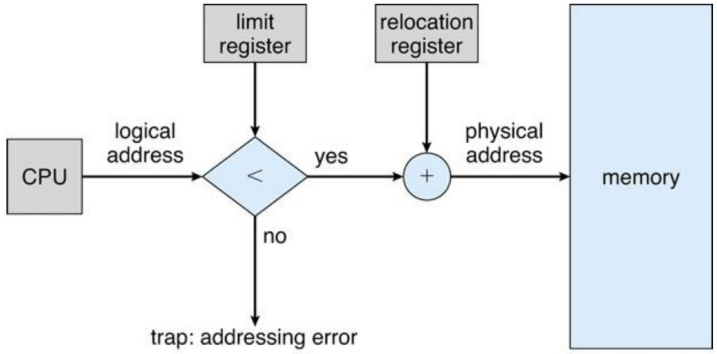
Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address
- In a MMU with relocation registers, the value of the register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with logical addresses; it never sees the real physical addresses

Dynamic Relocation using a Relocation Register



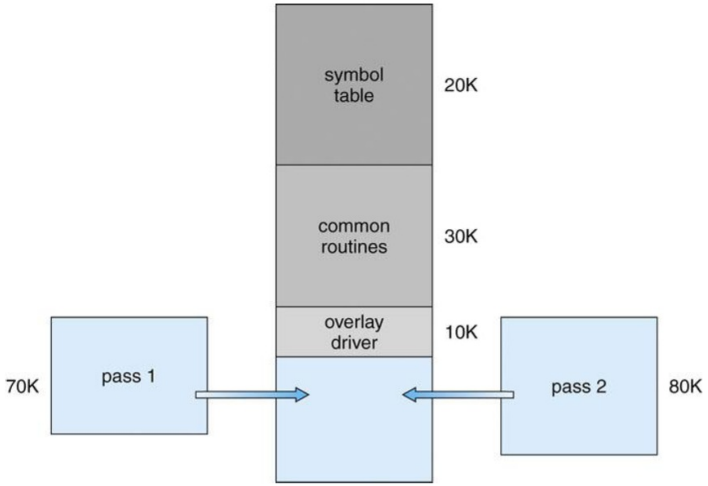
Hardware Support for Relocation and Limit Registers



Dynamic Loading

- Routine is not loaded until it is called
- Routine is is kept on disk in a relocatable load format
- Better memory-space utilization
  - Unused routine is never loaded
  - Overlays allow the loading of modules for the current phase of execution
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required implemented through program design (relocatable linking loader)

Overlays



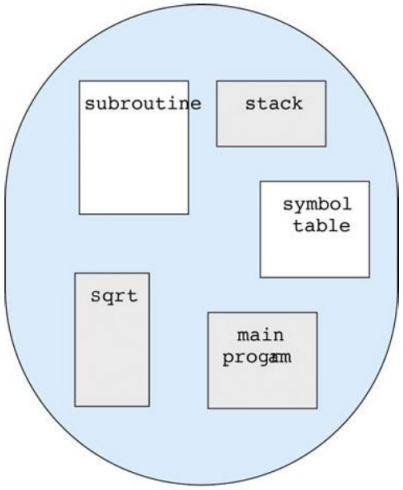
Dynamic Linking (Shared Libraries)

- Routine is not linked until it is called (linking is postponed)
- Small piece of code, stub, used to locate the appropriate memory-resident library routine (or how to load the library routine)
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system needed to check if routine is already in other processes' memory space
- Dynamic linking is particularly useful because it supports library updates

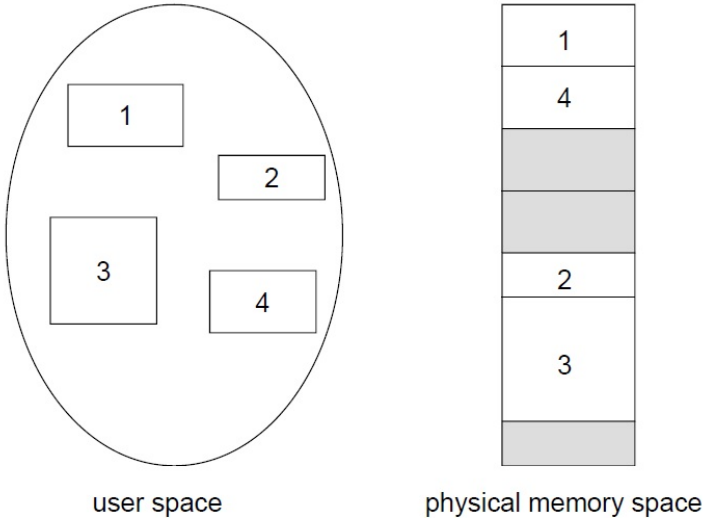
Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of segments
  - A segment is a logical unit such as:
    - main program
    - procedure
    - function
    - method
    - object
    - local/global variables
    - common block
    - stack
    - symbol table
    - arrays

Users View of a Program



Logical View of Segmentation



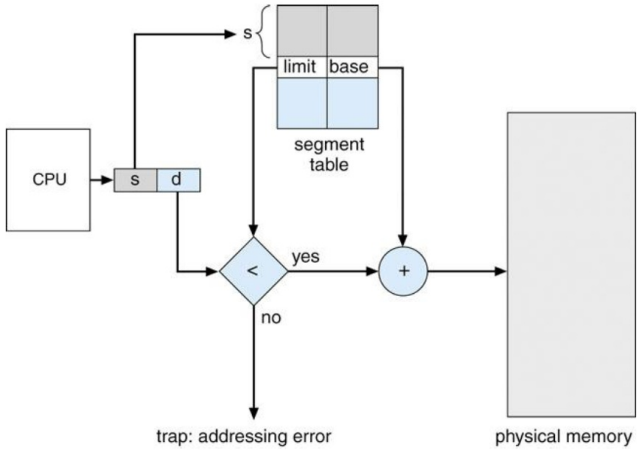
Segmentation Architecture

- Logical address consists of a two tuple:
  - $< SegmentNumber, Offset >$
  - offset = displacement (d)
- Segment table - maps two-dimensional physical addresses; each table entry has:
  - base - contains the starting physical address where the segments reside in memory
  - limit - specifies the length of the segment
- Segment-table base register (STBR) points to the segment tables location in memory
- Segment-table length register (STLR) indicates number of segments used by a program
  - segment number s is legal if s $<$ STLR

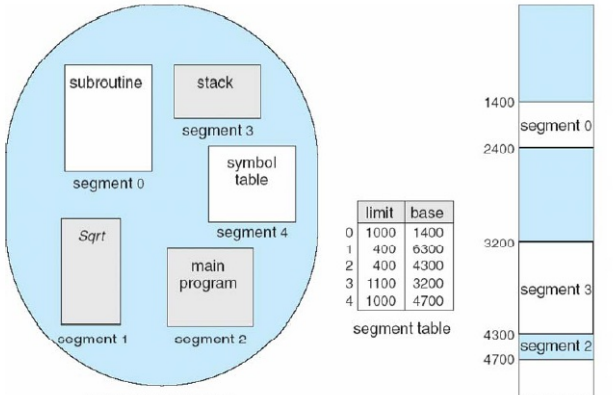
Segmentation Architecture II

- Protection
  - With each entry in segment table associate:
    - validation bit = 0 $\Rightarrow$ illegal segment
    - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem

Segmentation Hardware



Example of Segmentation



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

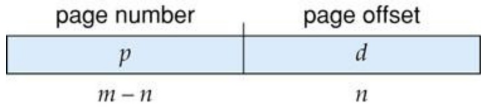
segment table

Paging

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Divide physical memory into fixed-sized blocks called frames (size is power of 2, between 512 bytes and 8,192 bytes)
- Divide logical memory into blocks of same size called pages
- Keep track of all free frames
- To run a program of size n pages, need to find n free frames and load program
- Set up a page table to translate logical to physical addresses
- Internal fragmentation (50% of page size on the overage)

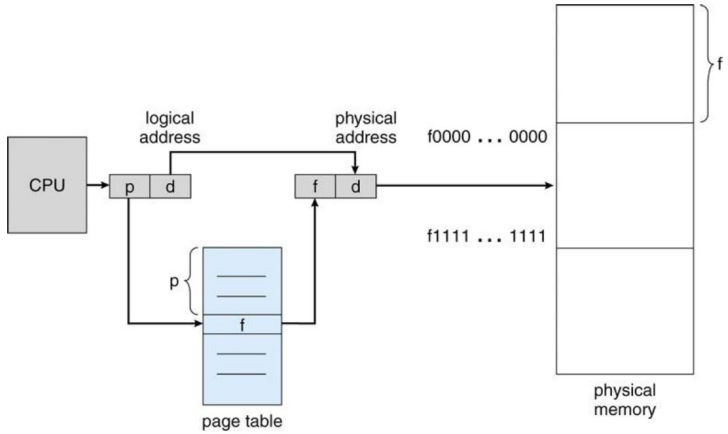
Address Translation Scheme

- Address generated by CPU 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) - combined with base address to define the physical memory address that is sent to the memory unit

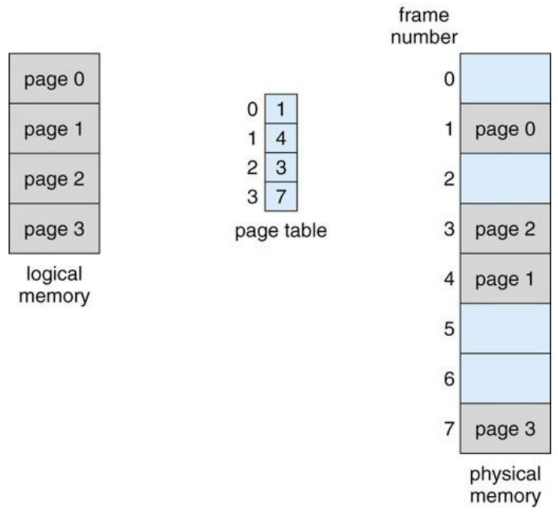


  - For given logical address space 2^m and page size 2^n

Paging Hardware

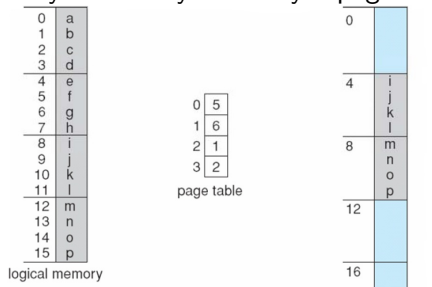


Paging Model of Logical and Physical Memory



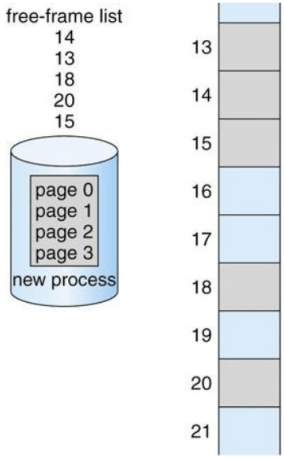
Paging Example

32-byte memory and 4-byte pages

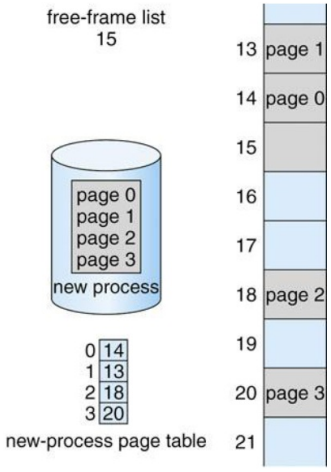


Free Frames

Before allocation



After allocation

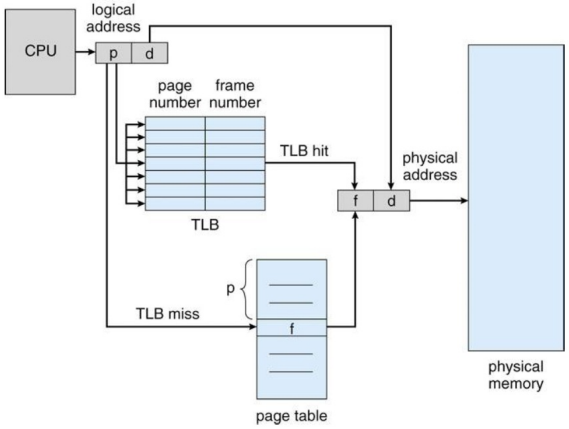


Implementation of Page Table

- Page table is kept in main memory
- Page-table base register (PTBR) points to the page table
- Page-table length register (PRLR) indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses, one for the page table and one for the data/instruction.
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache made of associative memory called translation look-aside buffers (TLBs)
- Some TLBs store address-space identifiers (ASIDs) in each TLB entry
  - Uniquely identifies each process to provide address-space protection for that process
  - Avoids TLB flush and reload at each address-space switch

Associative Memory

- Associative memory - parallel search for translation (p, d)
  - If p is in associative register, get frame # out
  - Otherwise get frame # from page table in memory

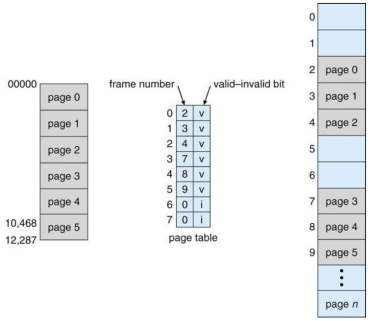


Effective Access Time

- Associative lookup takes τ time units (e.g., 1 ns)
- Assume memory cycle time is μ time units (e.g., 300 ns)
- Hit ratio α - percentage of times that a page number is found in the (e.g., 99 %) associative registers; ratio related to number of associative registers
- Effective Access Time (EAT)
  $EAT = (\tau + \mu) \cdot \alpha + (\tau + 2 \cdot \mu) \cdot (1 - \alpha) = \tau + 2 \cdot \mu - \mu \cdot \alpha$

Memory Protection with Valid/Invalid Bit

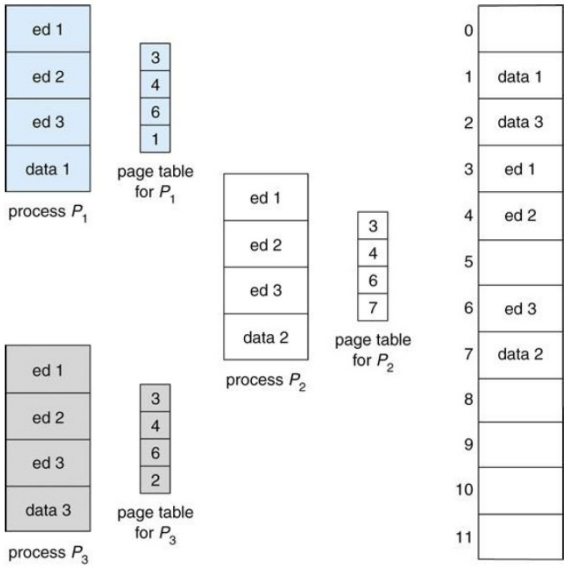
- Memory protection implemented by associating protection bit with each frame
- Valid-invalid bit attached to each entry in the page table:
  - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
  - "invalid" indicates that the page is not in the process' logical address space



Shared Pages

- Shared code
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  - Shared code must appear in same location in the logical address space of all processes (exception: position-independent code (PIC))
- Shared Data
  - Data with pointers must appear in same location in the logical address space of all processes
  - Data without pointers can appear anywhere in the logical address space
  - Synchronization is required for consistent read/write access
- Private code and data
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example

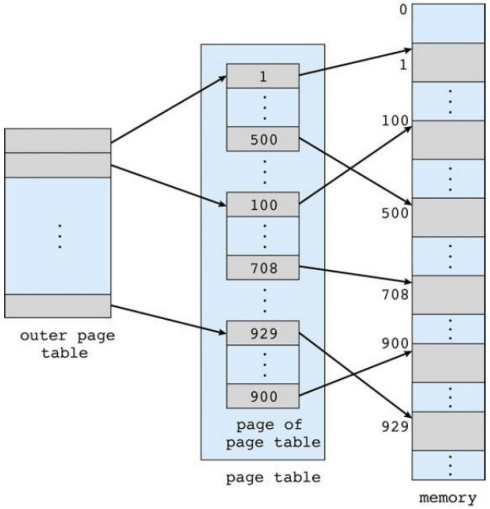


Structure of the Page Table

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

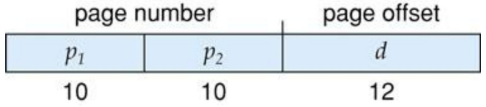
Hierarchical Page Tables

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table



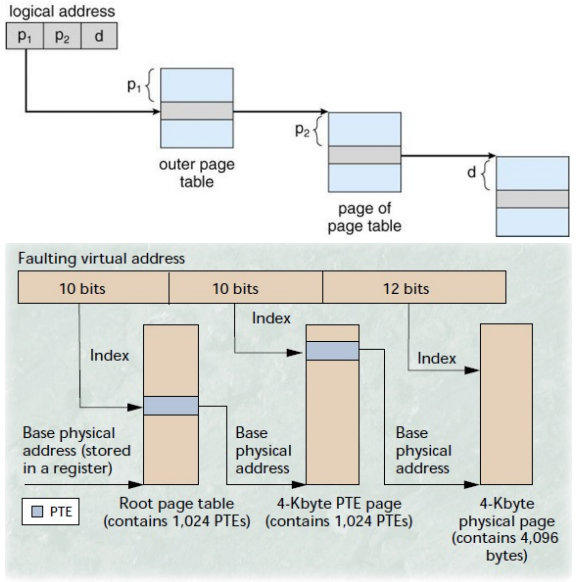
Two-Level Paging Example

- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page number consisting of 20 bits
  - a page offset consisting of 12 bits
- Since the page table is paged, the page number is further divided into:
  - a 10-bit page number
  - a 10-bit page displacement for level 2 page
- Thus, a logical address is as follows:

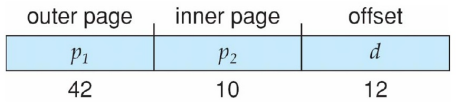


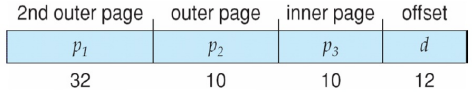
where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the outer page table

Top-Down Address-Translation Scheme

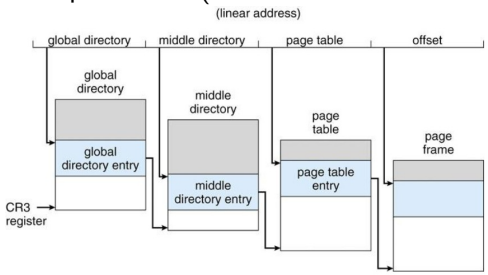


Top-Down Three-level Paging Scheme



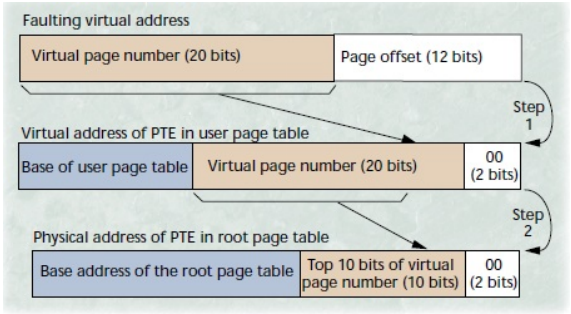


- Example: Linux (with 0 bit middle directory for 32 bit)



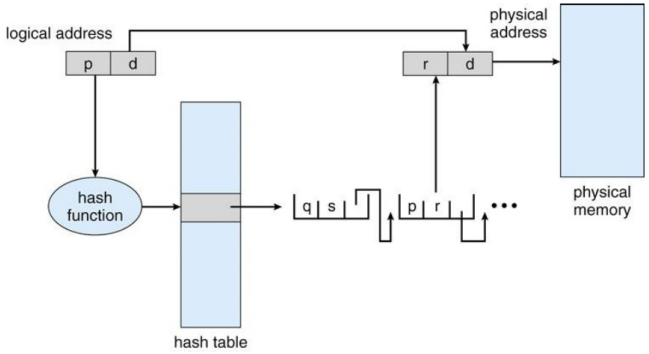
Bottom-Up Paging Scheme

- 4-Mbyte user page table is continuous in virtual space
- If mapping is not in TLB, load TLB from physical root page table
- Example: MIPS and Alpha



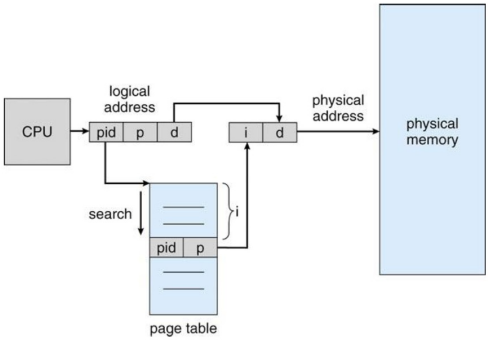
Hashed Page Tables

- Common in address spaces > 32 bits
- 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



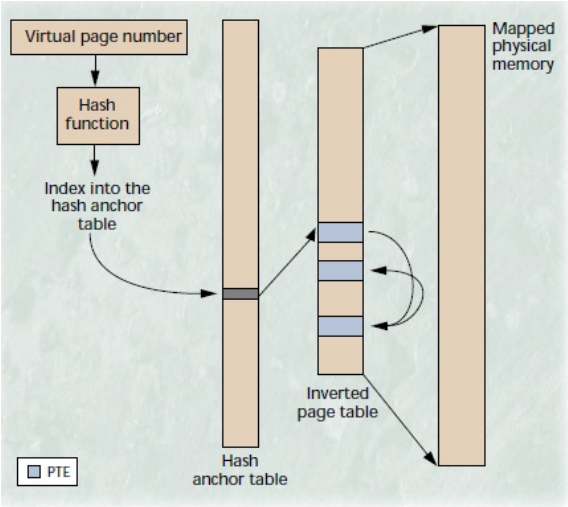
Linear Inverted Page Table

- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs

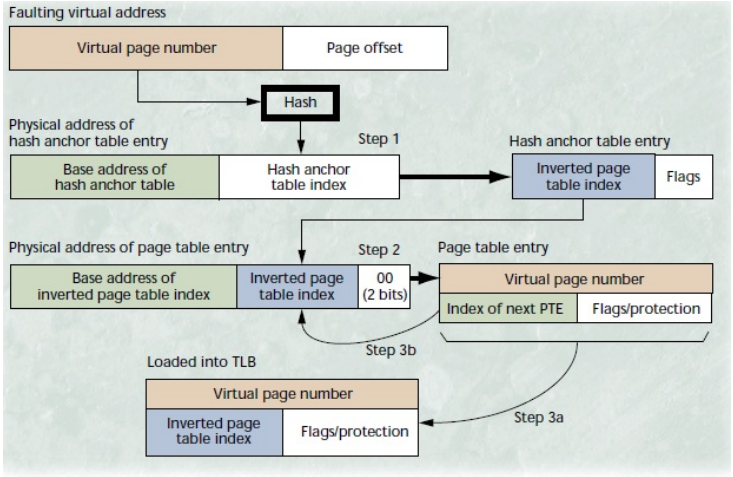


Hashed Inverted Page Table

- Use hash anchor table to limit the search to one - or at most a few - page-table entries (e.g., PPC, PA-RISC)



Hashed Inverted Page Table Lookup



VM Intro

- Problem Description
- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock