CAS CS 552
Intro to Operating Systems

Richard West

Memory Management

Memory Management

- Background
- Swapping
- Contiguous Memory Allocation
- Paging
- Structure of the Page Table
- Segmentation
- Example: The Intel Pentium

Objectives

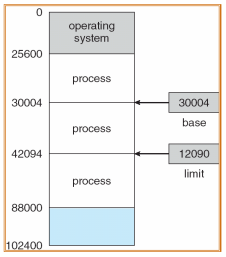
- To provide a detailed description of various ways of organizing memory hardware
- To discuss various memory-management techniques, including paging and segmentation
- To provide a detailed description of the Intel Pentium, which supports both pure segmentation and segmentation with paging

Background

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

Base and Limit Registers

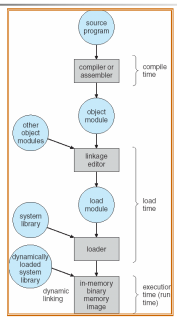
- One approach to define a logical address space is to use a pair of **base** and **limit** registers



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 mapping (virtual) address space of process to physical memory addresses

Multistep Processing of a User Program



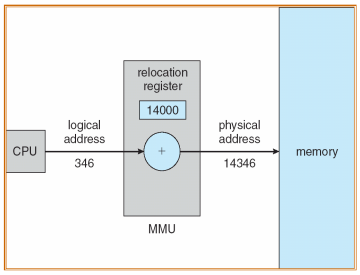
Logical vs. Physical Address Space

- 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)

- MMU = Hardware device that performs virtual to physical address translation
  - Example MMUs may use paging, segmentation, or relocation register techniques as part of address translation
- The user program deals with *logical* addresses; it never sees the *real* physical addresses

Dynamic relocation using a relocation register



Dynamic Loading

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- Useful when large amounts of code are needed to handle infrequently occurring cases

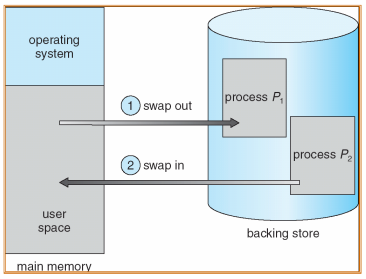
Dynamic Linking

- Linking postponed until execution time
- Small piece of code, *stub*, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system needed to check if routine is in processes' memory address space
- Dynamic linking is particularly useful for (shared) libraries

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** – e.g., disk large enough to accommodate copies of all memory images for all users
- **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
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

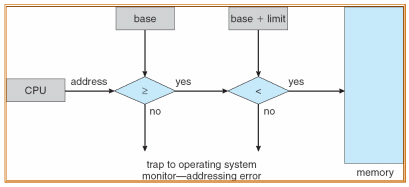
Schematic View of Swapping



Contiguous Allocation

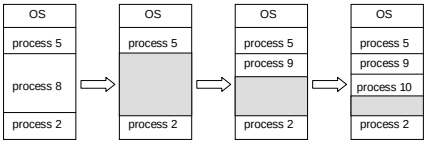
- Main (physical) memory usually split into two partitions:
  - Resident kernel, usually held in low memory
  - User processes, often held in high memory
- Techniques such as segmentation used to separate kernel from user-space
  - Segments have a **base** and **limit** on the range of memory addresses that can be accessed
    - Alternatively, hardware may provide separate base and limit registers

HW address protection with base and limit registers



Contiguous Allocation (Cont.)

- 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:
    a) allocated partitions b) 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
- **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the *largest* hole; must also search entire list
  - Produces the largest leftover hole

Fragmentation

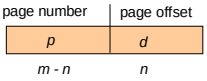
- **External Fragmentation** – total memory space that exists (external to processes), but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- Reduce external fragmentation 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
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers

Paging

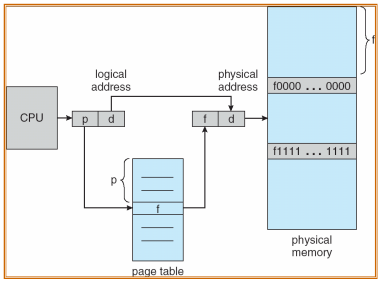
- Physical memory allocated to a process address space can be non-contiguous
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2; e.g., Intel IA-32 supports 4KB and 4MB page sizes)
- Divide logical memory into blocks of same size called **pages**
- Keep track of all 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 can still occur, due to unused memory within pages/frames

Address Translation Scheme

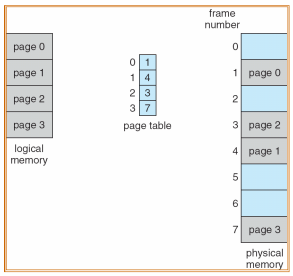
- With single-level paging, address generated by CPU is divided into:
  - **Page number (*p*)** – used as an index into a *page table* which contains base address of each page in physical memory
  - **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit
  - For given logical address space 2^m *and page size* 2^n



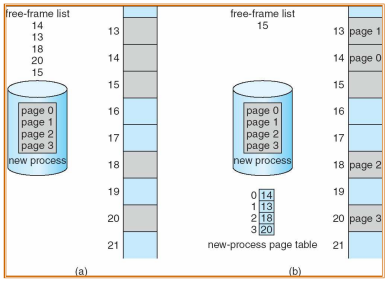
Paging Hardware



Paging Model of Logical and Physical Memory



Free Frames



Before allocation

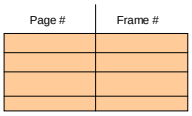
After allocation

Implementation of Page Tables

- Nowadays, each page table is usually kept in main memory
- **Page-table base register (PTBR)** points to the page table
- **Page-table length register (PTLR)** 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 called **associative memory** or **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

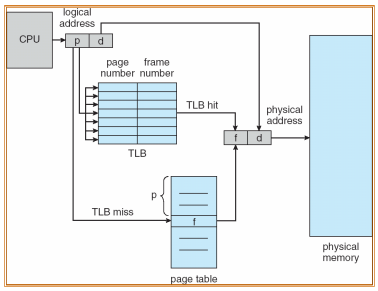
Associative Memory

- Associative memory – parallel search



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

Paging Hardware With TLB



Effective Access Time

- **Hit ratio** – percentage of times that a page number is found in associative memory
  - ratio related to number of associative registers
- Let:
  - hit ratio = α
  - memory access time = m
  - TLB search time = δ (assume same for hit or miss)
- **Effective Access Time** (EAT):
  - EAT = $\alpha(m+ \delta) + (1- \alpha)(2m+ \delta)$

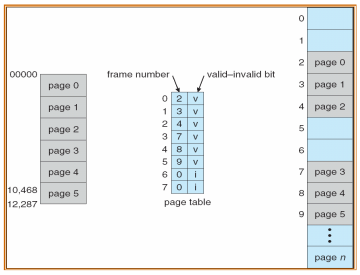
Example Access Time w/ TLB

- Let:
  - Hit ratio = 80%
  - One memory access takes 100nS
  - TLB search of associative registers = 20ns
  - For TLB hit, total access time = 120nS
  - For TLB miss, total access time = 220nS
  - EAT = 0.8x120 + 0.2x220 = 140nS
    - 40% slowdown over single memory access BUT
    - 30% improvement over accesses to memory always via page tables

Memory Protection

- 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
  - NOTE: Intel IA-32 has a *present bit* which serves the same purpose

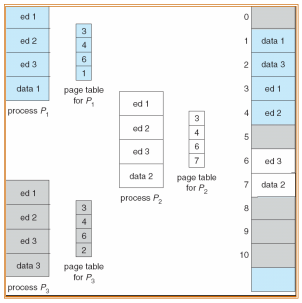
Valid/Invalid Page Table Entries



Shared Pages

- **Shared code**
  - One copy of read-only (reentrant) code shared among processes (e.g., code for an editor)
- **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



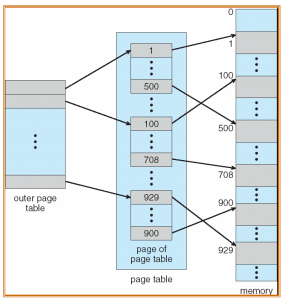
Structure of the Page Table

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

- Break up the logical address space into multiple page tables
  - Eliminates the need for large contiguous memory area for single-level page tables
  - e.g., 32-bit logical addresses, page-size=4KB
    - 2^{20} entries in page table; if each entry were 32-bits we'd need 4MB contiguous memory just for page table
- A simple technique is a two-level page table

Two-Level Page-Table Scheme

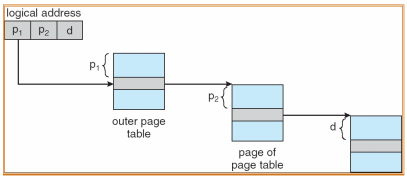


Two-Level Paging Example

- 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 offset
- Thus, a logical address is as follows:

page number		page offset
p_1	p_2	d
10	10	12

Address-Translation Scheme



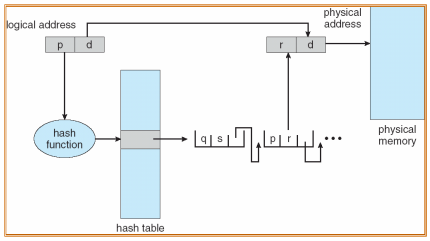
Example Three-level Paging Scheme (64-bit address spaces)

2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d
32	10	10	12

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

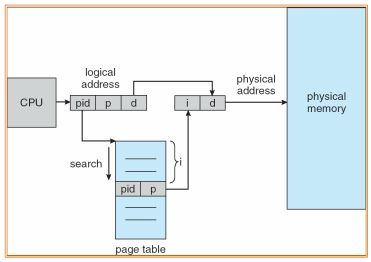
Hashed Page Table



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
- Use hash table to limit the search to one — or at most a few — page-table entries

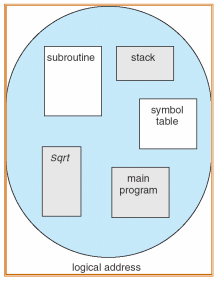
Inverted Page Table Architecture



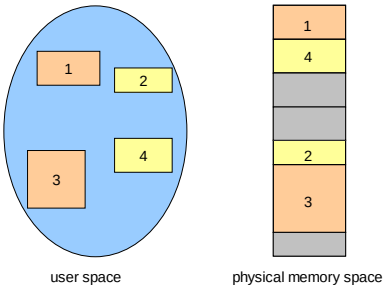
Segmentation

- Memory-management scheme that supports user's 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

User's View of a Program



Logical View of Segmentation



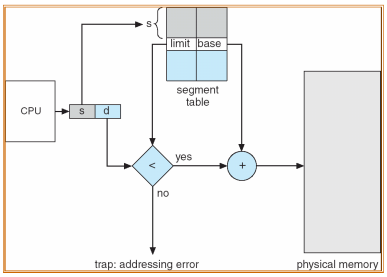
Segmentation Architecture

- Logical address consists of a two tuple:
  <segment-number : offset>
- **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 table's location in memory
  - **e.g., in Intel IA-32, LDTR and GDTR registers hold base physical addresses of local and global segment descriptor tables**

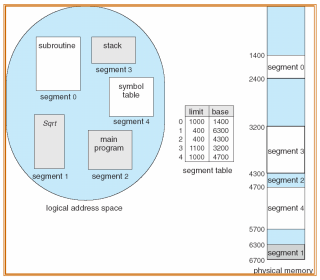
Segmentation Architecture (Cont.)

- Protection
  - With each entry in segment table associate:
    - validation bit = 0 ⇒ 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



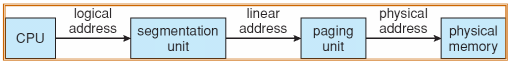
Example of Segmentation



Example: The Intel Pentium

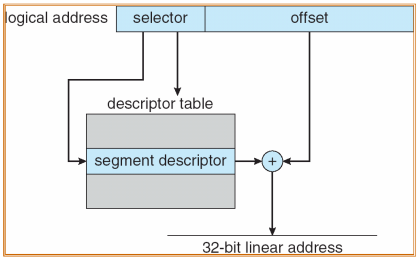
- Supports both segmentation and segmentation with paging
- CPU generates logical address
  - Given to segmentation unit
    - Which produces linear addresses
  - Linear address given to paging unit
    - Which generates physical address in main memory
    - MMU takes the form of a paging unit

Logical to Physical Address Translation in Pentium

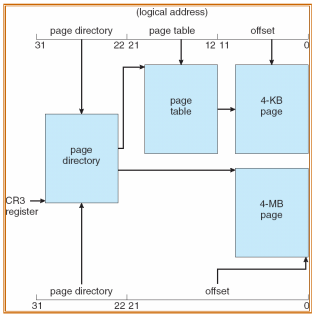


page number		page offset
p_1	p_2	d
10	10	12

Intel Pentium Segmentation



Pentium Paging Architecture



Linear Address in Linux

Broken into four parts:

global directory	middle directory	page table	offset

Three-level Paging in Linux