

CMSC 412

Spring 2007

Memory Management: Paging and Segmentation

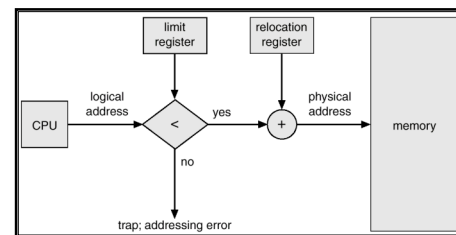
Announcements

- Reading:
 - Chapter 8 (except 8.1.2 - 8.1.5)
 - Chapter 9 (next time)

Logical vs. Physical Addresses

- 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.

Basic Relocation Hardware

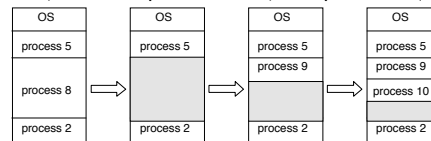


Contiguous Allocation

- Main memory usually in two partitions:
 - Resident operating system, usually held in low memory with interrupt vector.
 - User processes held in high memory.
- Single-partition allocation
 - Subdivide user-space into fixed-size chunks of memory. Each process gets a full chunk.
 - What if the process doesn't need that much?
 - What if it needs more?

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:
 - a) allocated partitions
 - b) free partitions (hole)



Dynamic Storage Allocation

- How to satisfy a request of size n from a list of free holes?
- Goal: efficient utilization of storage.
 - If we have n processes that require m bytes of memory where $nm < M$, where M is all available memory, we can fit all processes in memory at once.
- Goal: fast allocation times.
 - The time to find a hole of necessary size

Dynamic Storage Allocation

- **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.
- First-fit and best-fit better than worst-fit in terms of speed and storage utilization.

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.

Reducing Fragmentation

- Compaction
 - Shuffle memory contents to place all free memory together in one large block.
- Compaction is possible *only* if using relocatable logical addresses.
 - Why?
- I/O problem
 - Pin job in memory while it is doing I/O.
 - Or, do I/O only into (non-relocatable) OS buffers.

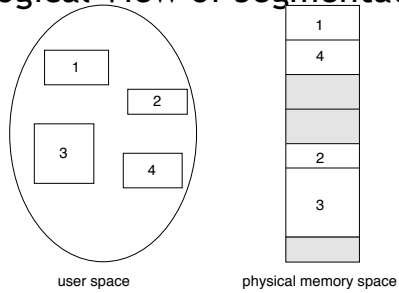
Non-Contiguous Allocation

- Can eliminate external fragmentation and improve storage utilization by allowing process memory to be *non-contiguous*.
 - Break the process memory into different chunks and allocate chunks in different parts of physical memory.

Segmentation

- Programs tend to be divided into different parts
 - Code
 - Data
 - Heap
 - Stack
- They can reside in separate *segments* that may be non-contiguous in physical memory.

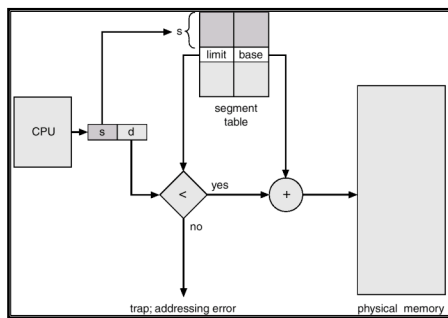
Logical View of Segmentation



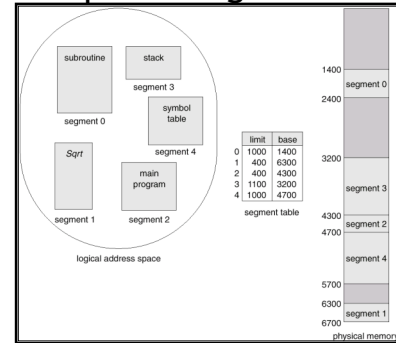
Segmentation Architecture

- Logical address consists of a two tuple:
 $\langle \text{segment-number}, \text{offset} \rangle$
- **Segment table** - maps segment # to
 - base - contains the starting physical address where the segments reside in memory.
 - limit - specifies the length of the segment.

Segmentation Hardware



Example of Segmentation



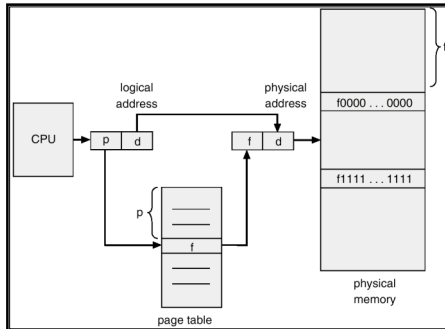
Paging

- Divide physical memory into fixed sized chunks called **frames** (pages), logical memory into same-sized **pages**
 - typical frames are 512 bytes to 64 Kbytes
 - When a process is to be executed, load the pages that are needed into memory.
- Have a map from pages to frames
 - Called the **page table**

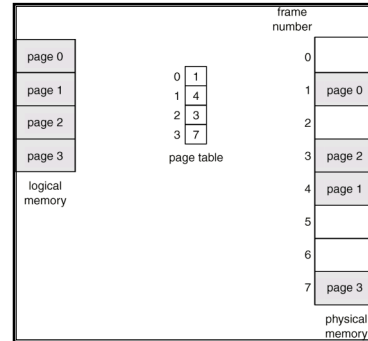
Address Translation Scheme

- Address generated by CPU is divided into:
 - **Page number** (p) - used as an index into the 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.
- Consider 32-bit addresses on Pentium:
 - 4096-byte pages (12 bits for the offset)
 - 20 bits for the page number

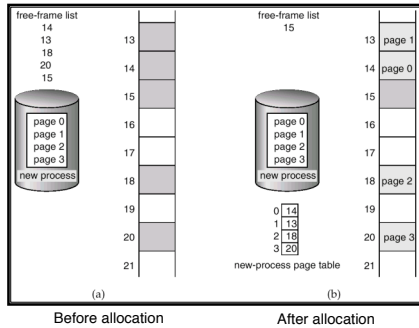
Address Translation



Paging Example



Free Frames



Paging Fragmentation

- If we have 4K pages, and a process requires 15K of memory, how many pages will we need?
- Suffer from *internal fragmentation*: last page likely not completely used
 - On average 50% of a page lost per process to internal fragmentation
- How does this impact our choice of page size?

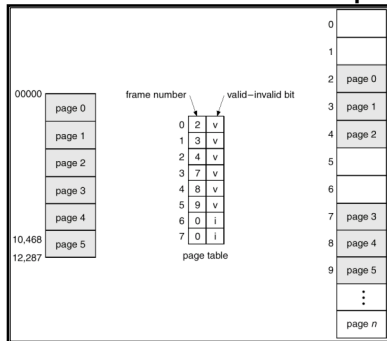
Implementation of Page Table

- Page table is kept in main memory.
- *Page-table base register (PTBR)* points to the page table.

Memory Protection

- Memory protection 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.

Valid/Invalid Bit Example



Problem: Page Table Size

- One page table can get very big
 - 2^{20} entries
 - for most programs, most items are empty
- Simple Solution
 - *Page-table length register* (PRLR) indicates size of the page table.
- But not as good for “sparse” address spaces

Hierarchical Page Tables

- Break up the logical address space into multiple page tables.
- Simplest case: a two-level page table.
 - Used on the Pentium. The outer table is called the *page directory*, which points to individual *page tables*. Each page table is itself a page.

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.

Two-Level Paging Example

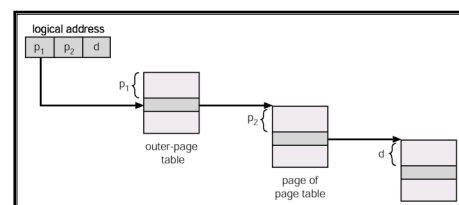
- Thus, a logical address is as follows:

page number		page offset
p_1	p_2	d
10	10	12

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.

Address-Translation Scheme

- Address-translation scheme for a two-level 32-bit paging architecture



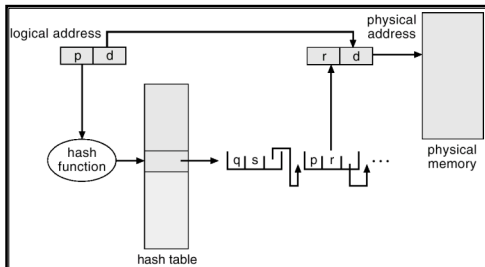
Problem: Still too big?

- What about 64 bit address spaces?
- Two-level page table:
 - Page num: 52 bits, or 2^{52} entries in page directory!
- More levels of hierarchy?
 - Further subdivide the page number.
 - 32-bit SPARC: three-level scheme.
 - 32-bit 68030: four-level scheme
- Try something different ...

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.

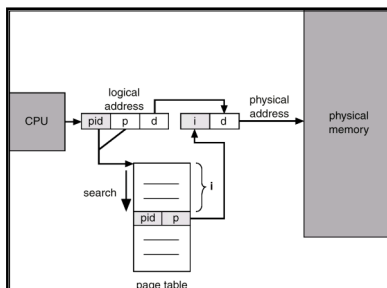
Hashed Page Table



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 the page table, but increases per-ref search time
 - Memory directly proportional to physical address space size
- Use hash table to limit the search to one – or at most a few – page-table entries.

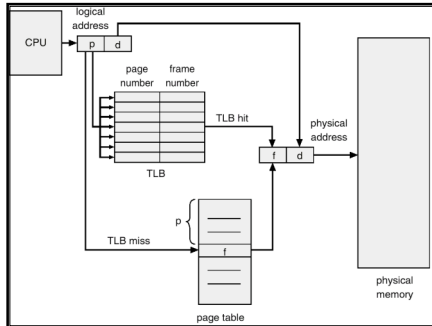
Inverted Page Table Architecture



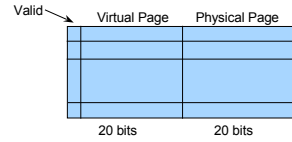
Problem: Address Lookup

- Every data/instruction access requires many memory accesses. At the least, 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)*
 - typically 16- 64 entries

Paging Hardware With TLB



Example: i386 TLB



Effective Access Time

- Associative Lookup = ϵ time units
- Memory access = 1 time unit
- Hit ratio = α
 - Fraction of time that a page number is found in the TLB.
- Effective Access Time (EAT)

$$\text{EAT} = (1 + \epsilon) \alpha + (2 + \epsilon)(1 - \alpha)$$

$$= 2 + \epsilon - \alpha \text{ time units}$$

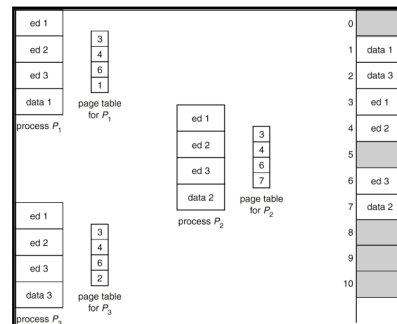
Super Pages

- TLB Entries
 - Tend to be limited in number
 - Can only refer to one page
- Idea
 - Create bigger pages
 - 4MB instead of 4KB
 - One TLB entry covers more memory
 - What's the tradeoff?

Shared Pages

- Shared code
 - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
 - May require contiguous range of logical addresses
- Private data
 - Each process keeps a separate copy of its own data.

Shared Pages Example



Sharing Writable Pages

- When writes occur, decide if processes share data
 - operating systems often implement *copy on write* - pages are shared until a process carries out a write
 - when a shared page is written, a new page frame is allocated
 - writing process owns the modified page
 - all other sharing processes own the original
 - processes use semaphores or other means to coordinate access

Sharing and Inverted PTs

- With the scheme just presented, each process's page table maps a different logical page to the same physical frame.
- When there is only one page table, indexed in part by process IDs, how might this work?

Paging and Segmentation

- The Pentium and other architectures combine these. On the Pentium
 - Logical addresses are mapped via segmentation to linear addresses
 - Linear addresses are mapped via paging to physical addresses
 - By default, only segmentation is used

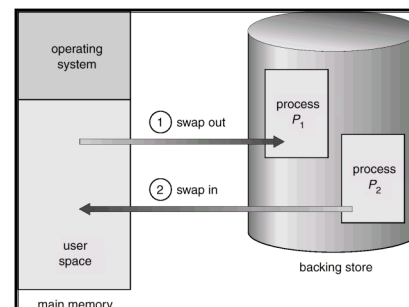
P&S: Protection Options

- How can we provide address space protection?
 - Using segmentation: protection enforced at the logical address level
 - Using paging: make segmentation a "no-op" and provide it at the linear address level
- What are the tradeoffs?
- Project 4 does the latter
 - More details in recitation slides

Swapping

- Main memory is big, but what if we run out?
- A process can be *swapped* temporarily out of memory to a *backing store*, and then brought back into memory for continued execution.
 - Backing store (disk) *bigger* than main memory
- Major part of swap time is transfer time; total transfer time is directly proportional to the *amount* of memory swapped.
 - But disk is *slower* than main memory
- Swapping used on UNIX, Linux, Windows, ...

Schematic View of Swapping



Virtual Memory

- We can integrate swapping and paging; called **virtual memory**
 - Pages can be swapped to and from disk, rather than entire processes
 - Presents the illusion of a larger physical memory
- Virtual addressing simplifies model
 - Swapping: can load pages into different addresses in physical memory
 - Addressing: different processes have their own address spaces, starting at 0, separate from other processes