BUCKNELL UNIVERSITY
Computer Science

CSCI 315 Operating Systems Design

Virtual Memory

Notice: The slides for this lecture have been largely based on those accompanying the textbook *Operating Systems Concepts with Java*, by Silberschatz, Galvin, and Gagne (2007). Many, if not all, of the illustrations contained in this presentation come from this source.

Logical vs. Physical Address Space

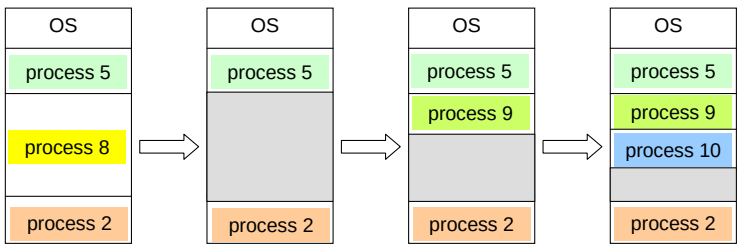
- The concept of a ***logical address space*** that is bound to a separate ***physical address space*** is central to proper memory management.
  - **Logical address** – generated by the CPU; also referred to as *virtual address*.
  - **Physical address** – address seen by the memory unit.
- Logical 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.

Contiguous Allocation

- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes then held in high memory.
- Single-partition allocation
  - Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data.
  - Relocation-register contains value of smallest physical address; limit register contains range of logical addresses – each logical address must be less than the limit register.

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 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.

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.
- **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

- 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 8192 bytes).
- Divide logical memory into blocks of same size called **pages** (we want to make page size equal to frame size).
- 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.

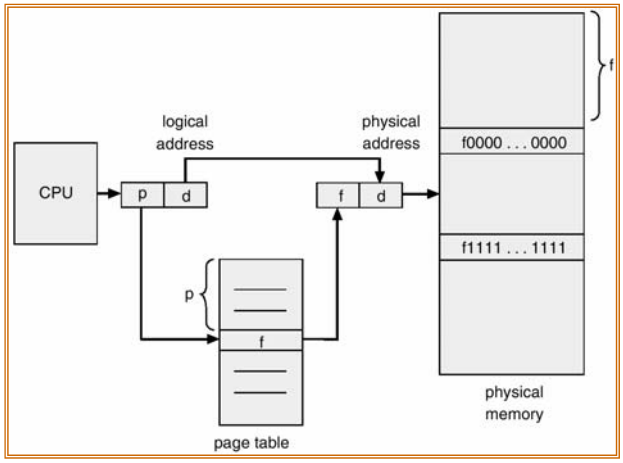
Address Translation Scheme

Address generated by CPU is divided into:

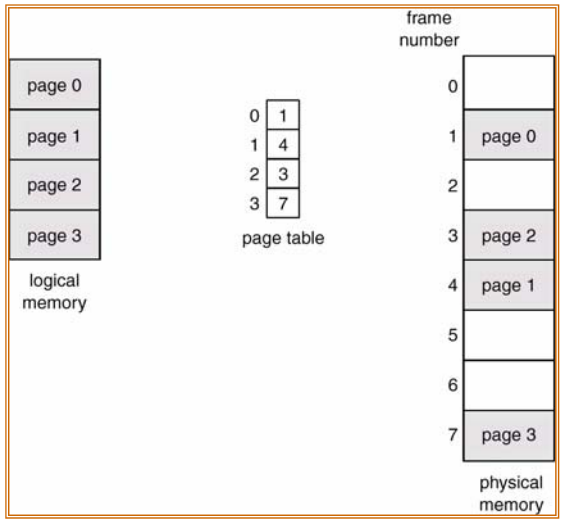
- ***Page number (p)*** – used as an index into a *page table* which contains base address of each page in physical memory.

- ***Page offset (d)*** – combined with base address to define the physical memory address that is sent to the memory unit.

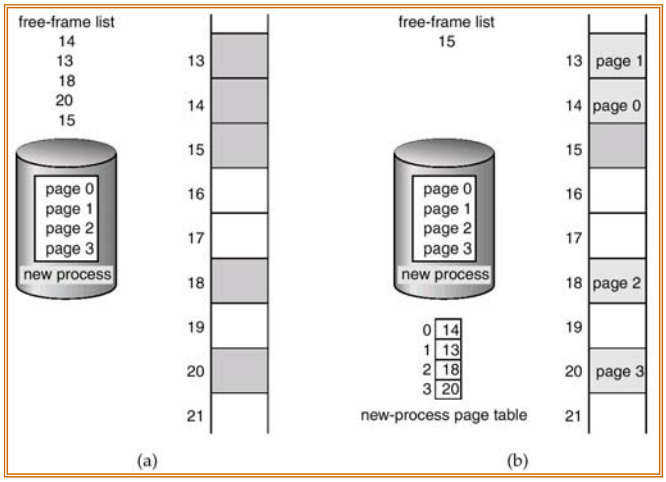
Address Translation Architecture



Paging Example



Free Frames



Before allocation

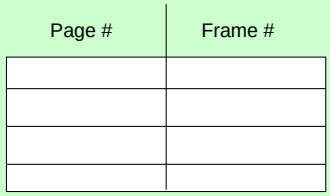
After allocation

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 called ***associative memory*** or ***translation look-aside buffers (TLBs).***

Associative Memory

Associative memory – parallel search

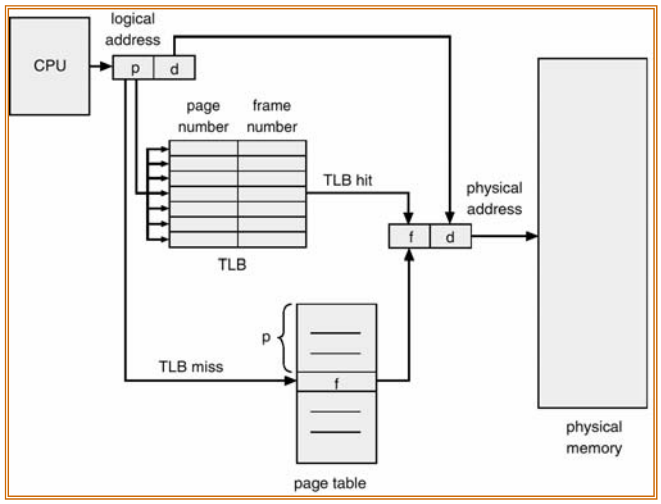


Address translation (A´, A´´)

- If A´ is in associative register, get frame # out.
- Otherwise get frame # from page table in memory

Associative memory is used to implement a TLB. Note that the TLB is nothing more than a special purpose **cache memory** to speed up access to the page table.

Paging Hardware With TLB



Effective Access Time

- **Associative Lookup** = ε time unit
- Assume memory cycle time is 1 microsecond
- **Hit ratio** – percentage of times that a page number is found in the associative registers; ration related to number of associative registers.
- Hit ratio = α
- **Effective Access Time (EAT)**

$$EAT = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha)$$
$$= 2 + \varepsilon - \alpha$$

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.

Hierarchical Page Tables

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

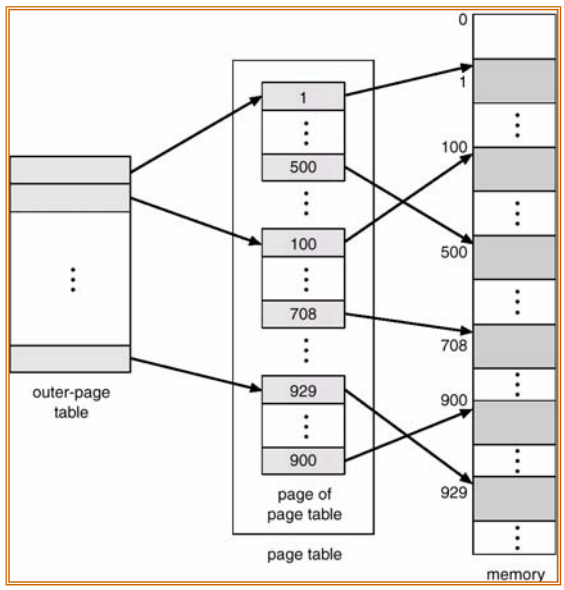
Two-Level Paging Example

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

page number		**page offset**
p_1	p_2	d
10	**10**	**12**

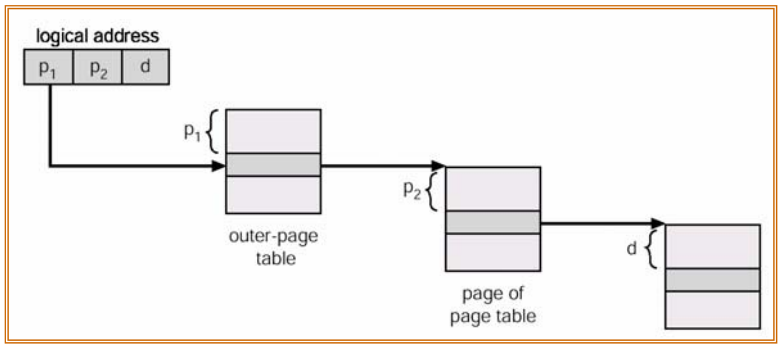
where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the outer page table.

Two-Level Page-Table Scheme



Address-Translation Scheme

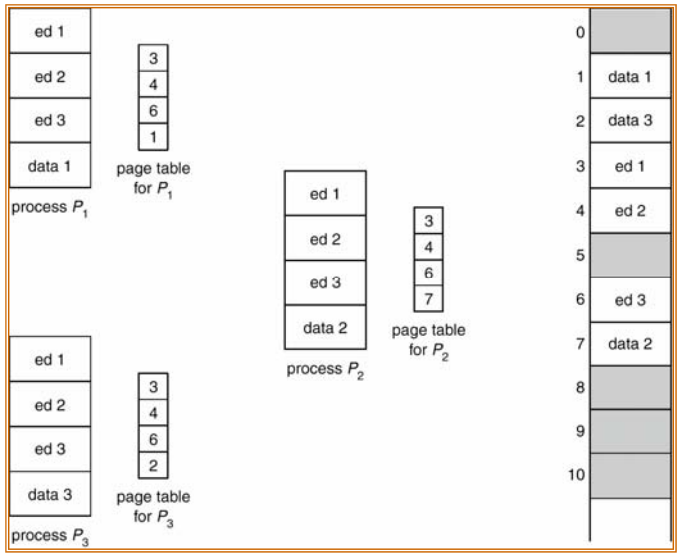
Address-translation scheme for a two-level 32-bit paging architecture:



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.
- 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.

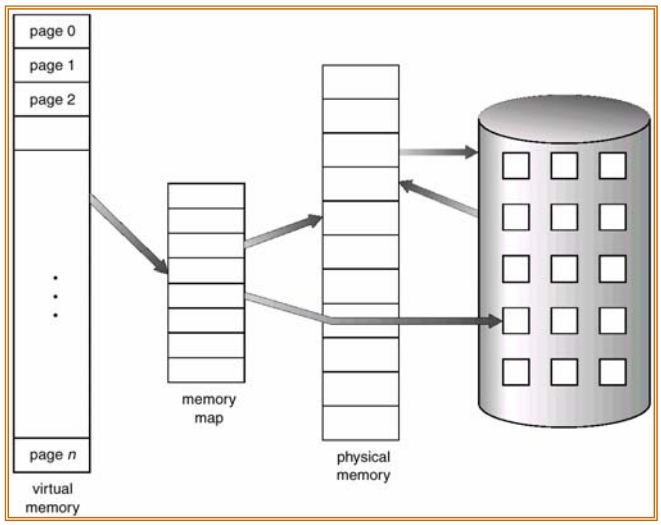
Shared Pages Example



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.
  - Allows for more efficient process creation.
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

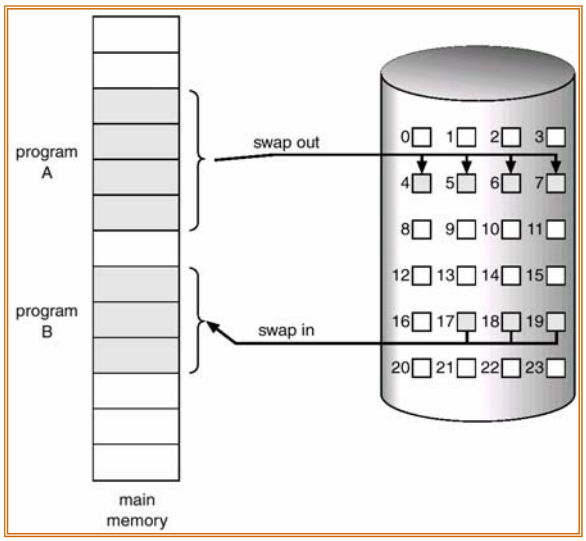
Virtual Memory
Larger than Physical Memory



Demand Paging

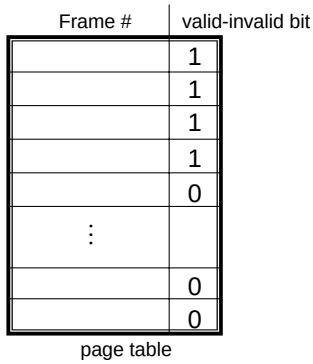
- Bring a page into memory only when it is needed.
  - Less I/O needed.
  - Less memory needed.
  - Faster response.
  - More users.
- Page is needed (there is a reference to it):
  - invalid reference $\Rightarrow$ abort.
  - not-in-memory $\Rightarrow$ bring to memory.

Transfer of a Paged Memory to Contiguous Disk Space



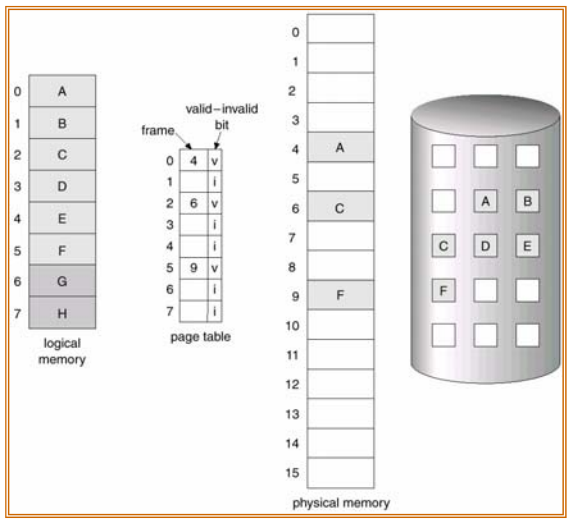
Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (1 ⇒ in-memory, 0 ⇒ not-in-memory)
- Initially valid–invalid but is set to 0 on all entries.
- Example of a page table snapshot.



- During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault.

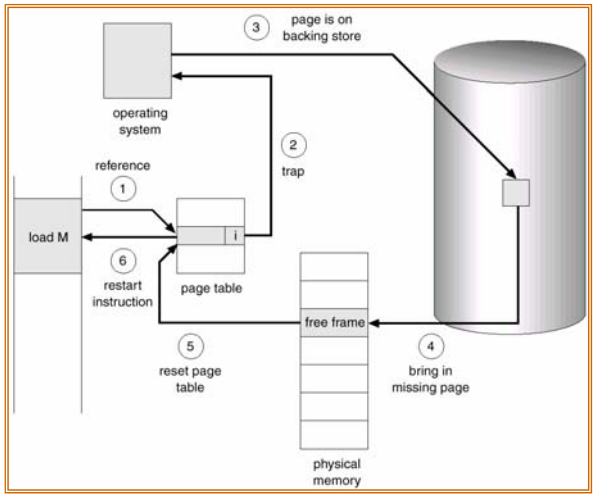
Page Table when some pages are not in Main Memory



Page Fault

- If there is ever a reference to a page, first reference will trap to OS ⇒ page fault.
- OS looks at page table to decide:
  - If it was an invalid reference ⇒ abort.
  - If it was a reference to a page that is not in memory, continue.
- Get an empty frame.
- Swap page into frame.
- Correct the page table and make validation bit = 1.
- Restart the instruction that caused the page fault.

Steps in Handling a Page Fault



What if there is no free frame?

- Page replacement – find some page in memory, that is not *"really"* in use and swap it out.
  - Must define an algorithm to select what page is replaced.
  - Performance: want an algorithm which will result in minimum number of page faults.
- The same page may be brought in and out of memory several times.