

Memory Management

1

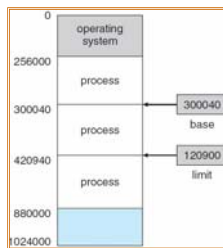
Background

- Program must be brought (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 (or less)
- Main memory can take many cycles
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

2

Base and Limit Registers

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



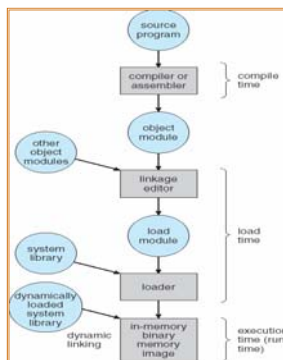
3

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
 - **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)

4

Multistep Processing of a User Program



5

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 address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme

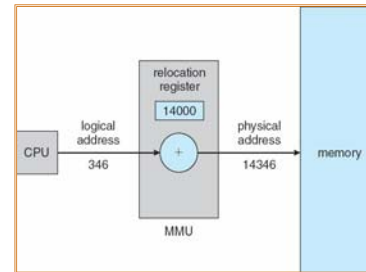
6

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address
- In MMU scheme, the value in the relocation (base) 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

7

Dynamic Relocation Using a Relocation Register



8

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
- No special support from the operating system is required, implemented through program design.

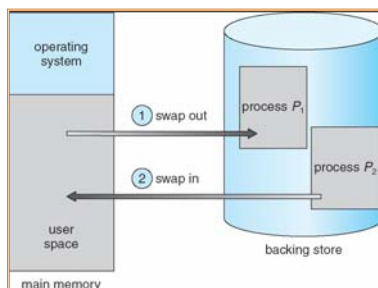
9

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** – 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
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

10

Schematic View of Swapping



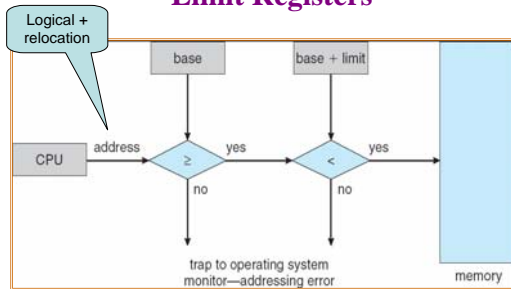
11

Contiguous Allocation

- Main memory usually divided into two partitions:
 - Resident operating system, usually held in low memory.
 - User processes then held in high memory.
- Relocation 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 – each logical address must be less than the limit register.
 - MMU maps logical address *dynamically*.

12

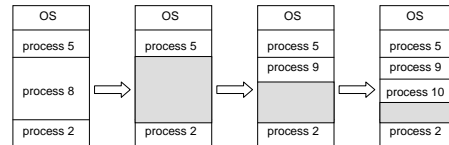
HW Address Protection with Base and Limit Registers



13

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)



14

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
 - 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

15

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

16

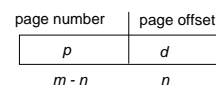
Paging

- 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).
- 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.

17

Address Translation Scheme

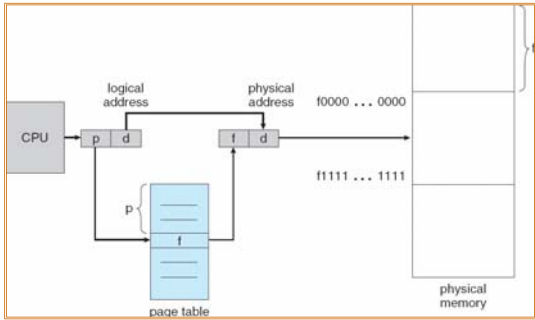
- Address generated by CPU is divided into:
 - **Page number (p)** – used as an index into a *page table* which contains base address of each page in physical memory
 - **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit



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

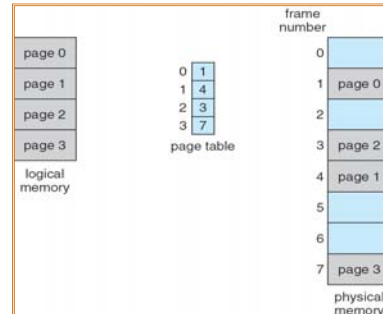
18

Paging Hardware



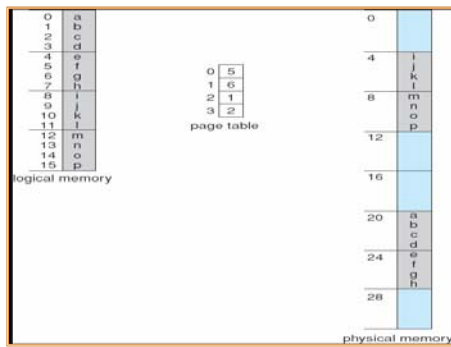
19

Paging Model of Logical and Physical Memory



20

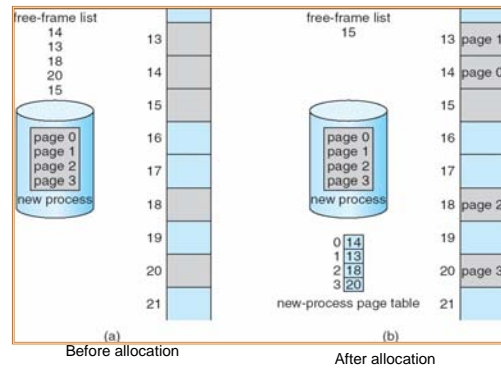
Paging Example



32-byte memory and 4-byte pages

21

Free Frames



Before allocation

After allocation

22

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)**
- Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry – uniquely identifies each process to provide address-space protection for that process

23

Associative Memory

- Associative memory – parallel search

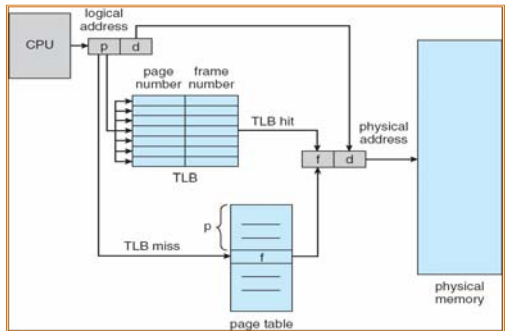
Page #	Frame #

Address translation (p, d)

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

24

Paging Hardware With TLB



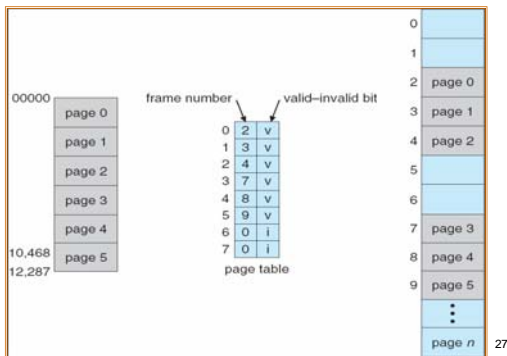
25

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.

26

Valid (v) or Invalid (i) Bit



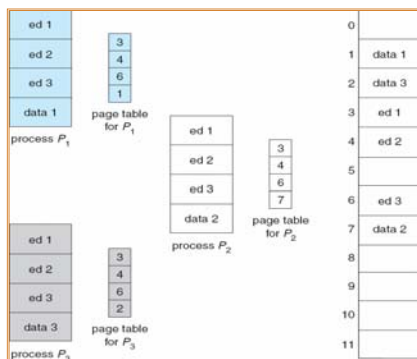
27

Shared Pages

- **Shared code**
 - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
 - Each page table maps onto the same physical copy of the shared code.
- **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.

28

Shared Pages Example



29

Structure of the Page Table

- As the number of processes increases, the percentage of memory devoted to page tables also increases.
- The following structures solved this problem:
 - Hierarchical Paging
 - Hashed Page Tables
 - Inverted Page Tables

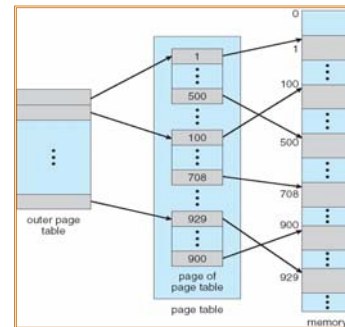
30

Hierarchical Page Tables

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

31

Two-Level Page-Table Scheme



32

Two-Level Paging Example

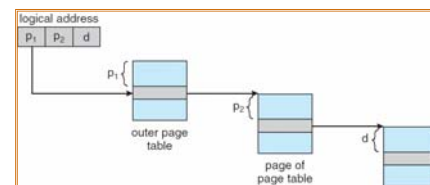
- A logical address (on 32-bit machine with 1K page size) is divided into:
 - a page number consisting of 22 bits
 - a page offset consisting of 10 bits
- Since the page table is paged, the page number is divided into:
 - a 12-bit page number
 - a 10-bit page offset
- Thus, a logical address is as follows:

page number		page offset
p_1	p_2	d
12	10	10

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

33

Address-Translation Scheme



34

Three-level Paging Scheme

outer page	inner page	offset
p_1	p_2	d
42	10	12

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

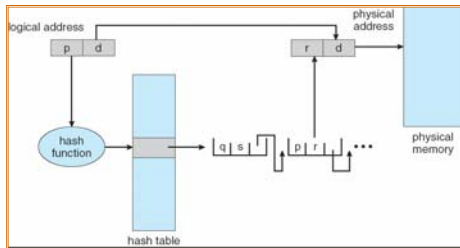
35

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.

36

Hashed Page Table



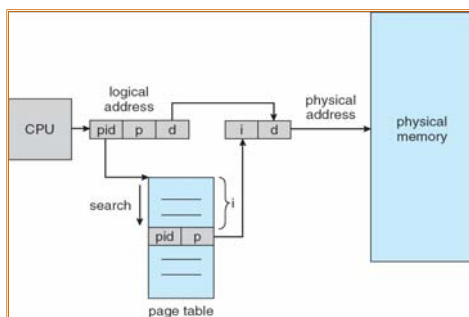
37

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.

38

Inverted Page Table Architecture



39

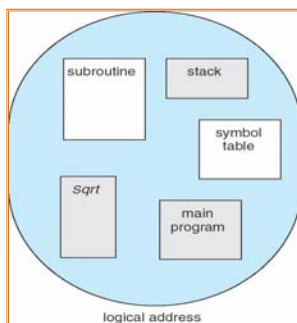
Segmentation

- 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 variables, global variables,
 common block,
 stack,
 symbol table, arrays

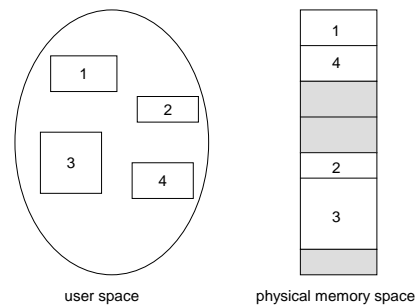
40

User's View of a Program



41

Logical View of Segmentation



42

Segmentation Architecture

- Logical address consists of a two tuple:
 $\langle \text{segment-number}, \text{offset} \rangle$,
- **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
- **Segment-table length register (STLR)** indicates number of segments used by a program; segment number s is legal if $s < \text{STLR}$

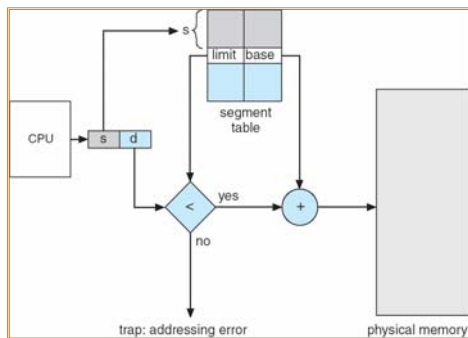
43

Segmentation Architecture - Cont.

- 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.

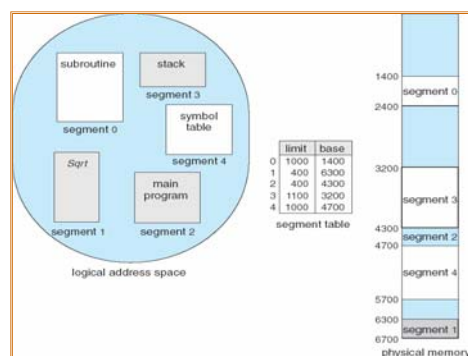
44

Segmentation Hardware



45

Example of Segmentation



46