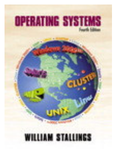
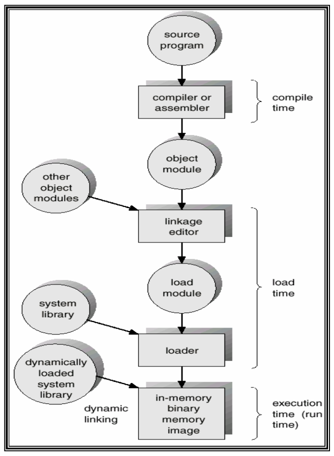
Memory Management



Chapter 7, Livro do William Stallings
Chapter 9, Livro do Silberschatz

SISTEMAS OPERATIVOS, 1º Semestre, 2004-2005



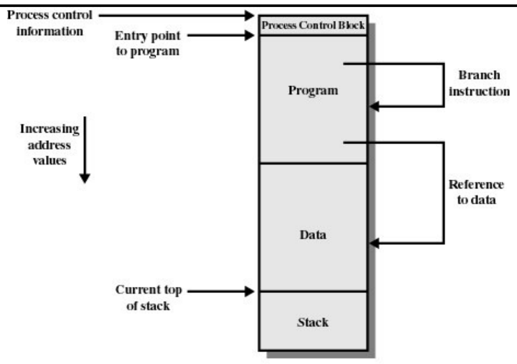


Figure 7.1 Addressing Requirements for a Process

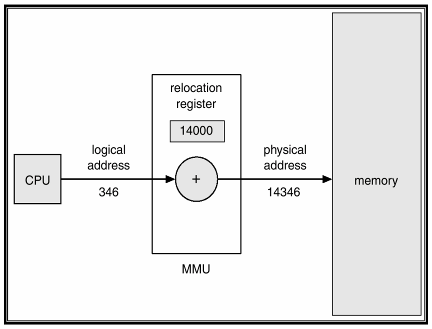
Memory Management

- The challenge is:
  - Sub-divide the memory to accommodate multiple processes.
- Memory needs to be allocated efficiently to pack as many processes into memory as possible.

Logical vs Physical Addresses

- **Logical address** – generated by the CPU; also referred to as *virtual address*.
- **Physical address** – address seen by the memory unit.

Dynamic Relocation



MMU

- Hardware device that maps virtual to physical address.
- In MMU scheme, the value in the relocation 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**.

Memory Relocation

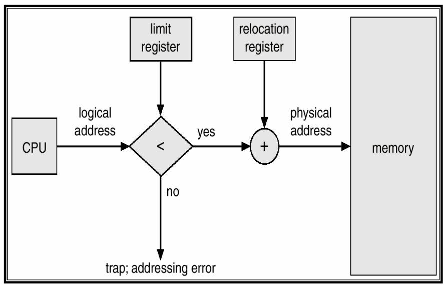
- **Relocation**
  - Programmer does not know where the program will be placed in memory when it is executed.
  - While the program is executing, it may be swapped to disk and returned to main memory at a different location (**relocated**).
  - Memory references must be translated in the code to actual physical memory address.

Memory Protection

- **Protection**
  - Processes should not be able to reference memory locations in another process without permission.
  - Impossible to check absolute addresses in programs since the program could be relocated.
  - Must be checked during execution
    - Operating system cannot anticipate all of the memory references a program will make.

Mechanisms that support relocation also support protection (hardware)

Memory Protection



Relocation-register: used to protect user processes from each other. Relocation register contains the value of the smallest physical address;
Limit register contains range of logical addresses – each logical address must be less than the limit register.

Memory Sharing

- **Sharing**
  - Allow several processes to access the same portion of memory.
  - Better to allow each process access to the same copy of the program rather than have their own separate copy.

Logical Organization

- **Logical Organization**
  - Programs are written in modules.
  - Modules can be written and compiled independently.
  - Different degrees of protection given to modules (read-only, execute-only).
  - Share modules.

Physical Organization

- **Physical Organization**
  - Memory available for a process may be insufficient
    - Overlaying allows various modules to be assigned the same region of memory
  - Programmer does not know how much space will be available.

Jump to slide 18

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.

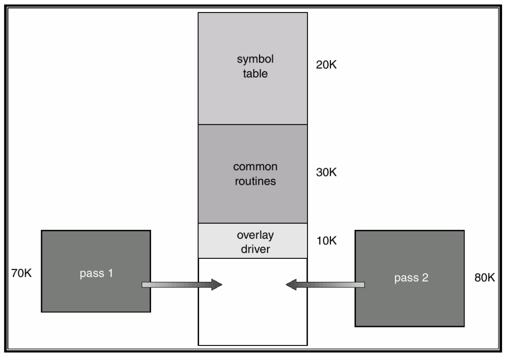
Dynamic Linking

- 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 needs to check if routine is in processes' memory address.
- Dynamic linking is particularly useful for libraries.

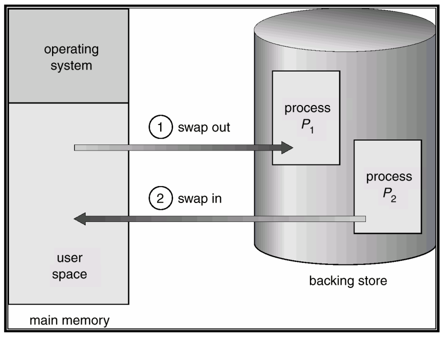
Memory Overlay

- Keep in memory only those instructions and data that are needed at any given time.
- Needed when process is larger than amount of memory allocated to it.
- Implemented by user, no special support needed from operating system.
- Programming design of overlay structure is complex

Memory Overlay



Swapping



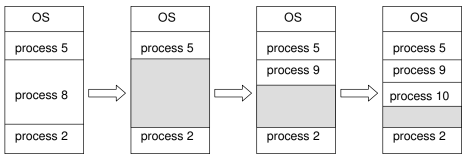
Swapping

- A process can be *swapped* temporarily out of memory to disk, and then brought back into memory for continued execution.
- Major part of swap time is transfer time; total transfer time is directly proportional to the *amount* of memory swapped.
- Example:

disk: transfer rate	5Mb/sec
memory to swap:	1 Mb
Time it takes to swap out:	1/5 sec = 200 msec
Average disk latency:	8 msec
Time to swap-out/swap-in:	416 msec

Memory 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)



Memory Management Techniques

1- Fixed Partitioning
2- Dynamic Partitioning
3- Buddy System
4- Paging
5- Segmentation

1- Fixed Partitioning

1-Fixed Partitioning

- **Equal-size partitions**
  – any process whose size is less than or equal to the partition size can be loaded into an available partition.
  – if all partitions are full, the operating system can swap a process out of a partition.
  – a program may not fit in a partition. The programmer must design the program with overlays.

Fixed Partitioning

- Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This is called **internal fragmentation.**

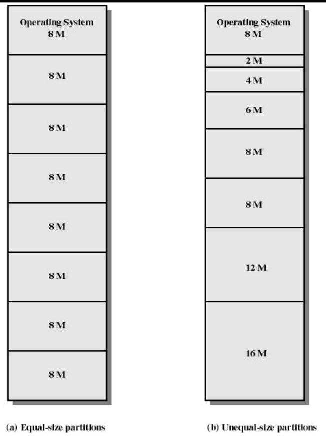


Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

Placement Algorithm with Partitions

- **Equal-size partitions**
  - because all partitions are of equal size, it does not matter which partition is used
- **Unequal-size partitions**
  - can assign each process to the smallest partition within which it will fit
  - queue for each partition
  - processes are assigned in such a way as to minimize wasted memory within a partition

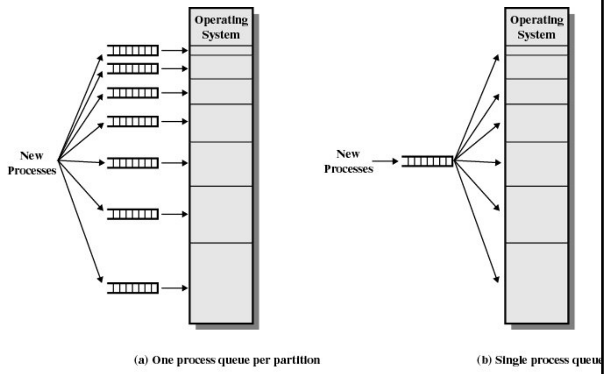


Figure 7.3 Memory Assignment for Fixed Partitioning

2- Dynamic Partitioning

2-Dynamic Partitioning

- Partitions are of variable length and number.
- Process is allocated exactly as much memory as required.
- Eventually get holes in the memory. This is called **external fragmentation**.
- Must use **compaction** to shift processes so they are contiguous and all free memory is in one block.

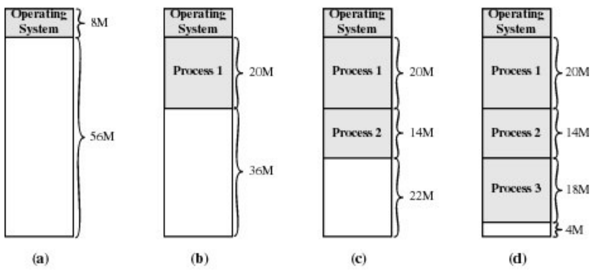


Figure 7.4 The Effect of Dynamic Partitioning

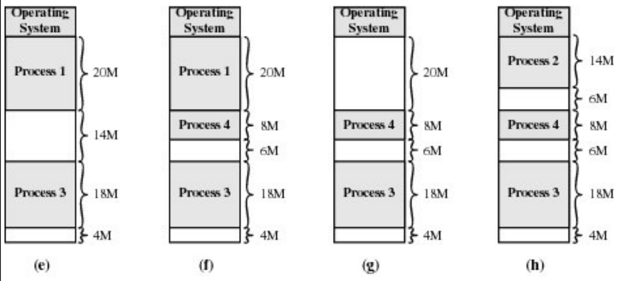


Figure 7.4 The Effect of Dynamic Partitioning

Memory Compaction

- To merge the memory blocks and reduce memory holes caused by external fragmentation.
- Time-consuming task....

Dynamic Partitioning Placement Algorithms

- Operating system must decide which free block to allocate to a process.
- Four algorithms:
  Best-Fit; First-Fit; Next-Fit; Worst-Fit

Dynamic Partitioning Placement Algorithm

- **Best-fit algorithm**
  - Chooses the block that is closest in size to the request.
  - Worst performance.
  - Since smallest block is found for process, the smallest amount of fragmentation is left memory compaction must be done more often.

Dynamic Partitioning Placement Algorithm

- **Worst-fit algorithm**
  - Chooses the largest available block.
  - The fragmentation area can be used for another request.

Dynamic Partitioning Placement Algorithm

- **First-fit algorithm**
  - Begins to scan memory from the beginning and chooses the next available block that is largest enough.
  - Fastest
  - May have many process loaded in the front end of memory that must be searched over when trying to find a free block.

Analysis of First-Fit Algorithm:

For every N allocated blocks, there are another 0.5N blocks that will be lost due to fragmentation -> 1/3 of memory will be unusable...

Dynamic Partitioning Placement Algorithm

- **Next-fit**
  - Begins to scan memory from the location of the last placement and chooses the next available block that is largest enough.

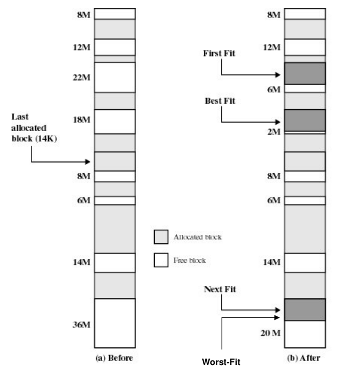


Figure 7.5 Example Memory Configuration Before and After Allocation of 16 Mbyte Block

3-Buddy-System

3- Buddy System

- Entire space available is treated as a single block of $2^{U.}$
- If a request of size s such that $2^{U-1} < s <= 2^{U}$, entire block is allocated.
  - Otherwise block is split into two equal buddies.
  - Process continues until smallest block greater than or equal to s is generated.

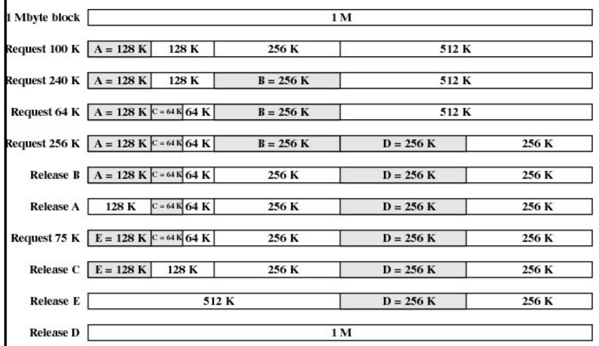

1 Mbyte block | 1 M
Request 100 K | A = 128 K | 128 K | 256 K | 512 K
Request 240 K | A = 128 K | 128 K | B = 256 K | 512 K
Request 64 K | A = 128 K | C = 64 K | 64 K | B = 256 K | 512 K
Request 256 K | A = 128 K | C = 64 K | 64 K | B = 256 K | D = 256 K | 256 K
Release B | A = 128 K | C = 64 K | 64 K | 256 K | D = 256 K | 256 K
Release A | 128 K | C = 64 K | 64 K | 256 K | D = 256 K | 256 K
Request 75 K | E = 128 K | C = 64 K | 64 K | 256 K | D = 256 K | 256 K
Release C | E = 128 K | 128 K | 256 K | D = 256 K | 256 K
Release E | 512 K | D = 256 K | 256 K
Release D | 1 M

Figure 7.6 Example of Buddy System

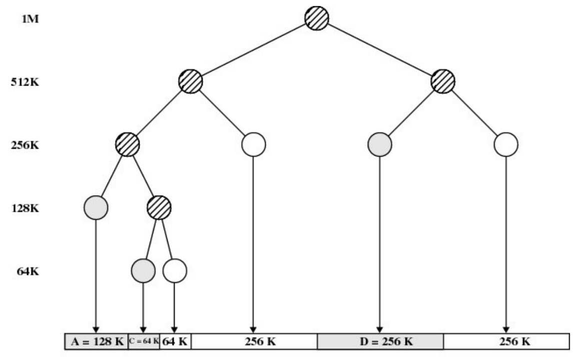


Figure 7.7 Tree Representation of Buddy System

4- Paging

4- Paging

- Partition memory into small equal-size chunks and divide each process into the same size chunks.
- The chunks of a process are called **pages** and chunks of memory are called **frames**.
- Operating system maintains a **page table** for each process
  - contains the frame location for each page in the process.
  - memory address consist of a page number and offset within the page.

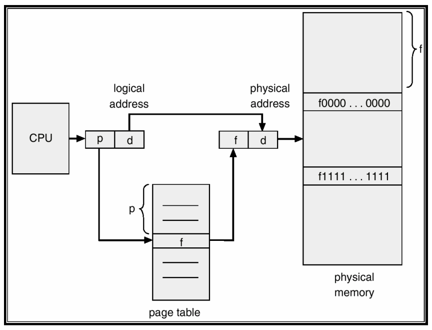
Paging

- OS should 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

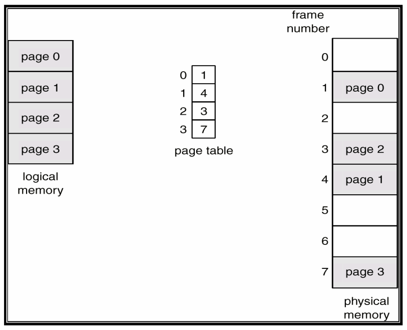
- Address generated by CPU is divided into:
  - **Page number (p)** – used as an index into a *page table* which contains base address of each page in physical memory.
  - **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit.

Paging: Address Translation

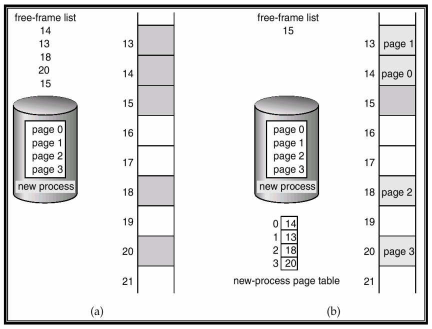


Page number (p) **– used as an index into a *page table* which contains base address of each page in physical memory.**
Page offset (d) **– combined with base address to define the physical memory address that is sent to the memory unit.**

Paging Example



Free Frames



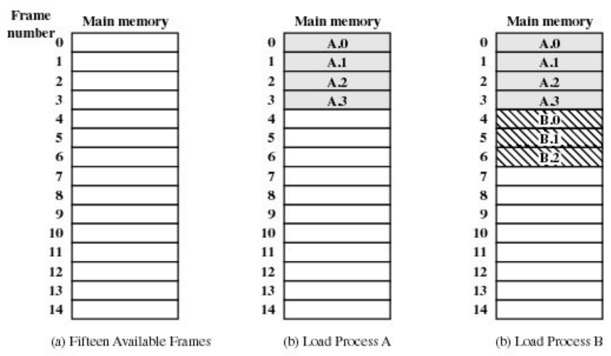


Figure 7.9 Assignment of Process Pages to Free Frames

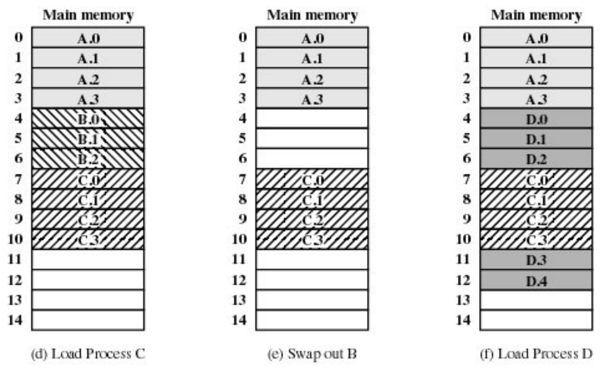


Figure 7.9 Assignment of Process Pages to Free Frames

Page Tables for Example

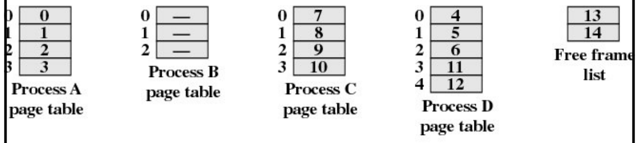
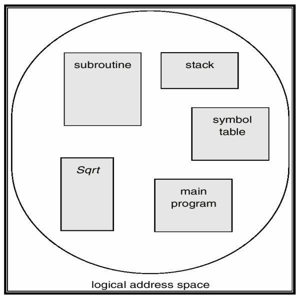


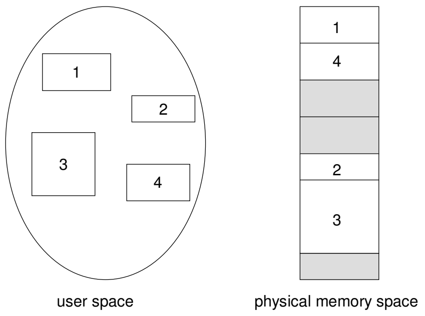
Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

5- Segmentation

User's View of a Program



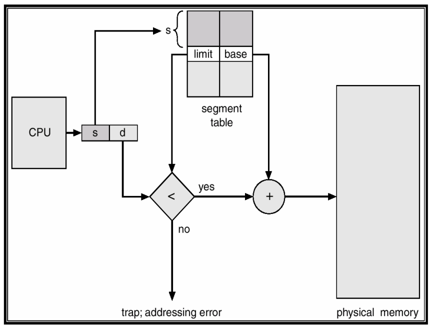
Logical View of Segmentation



5- Segmentation

- All segments of all programs do not have to be of the same length.
- There is a maximum segment length.
- Addressing consist of two parts:
  - a segment number and an offset.
- Since segments are not equal, segmentation is similar to dynamic partitioning.

Segmentation Hardware

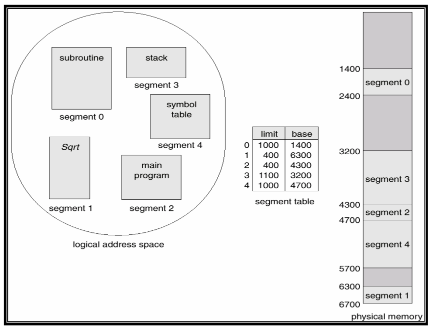


Segmentation Architecture

- 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.
- *Segment-table length register (STLR)* indicates number of segments used by a program;

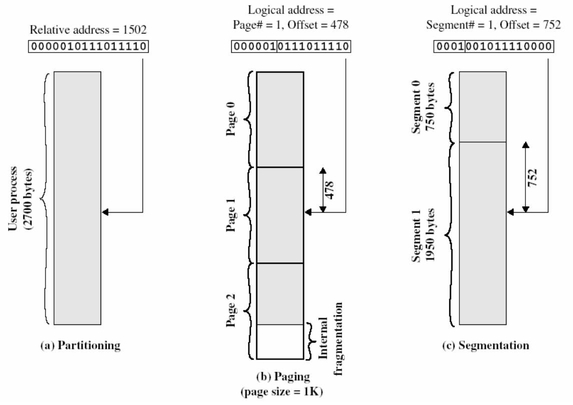
  segment number *s* is legal if *s* < STLR.

Example of Segmentation



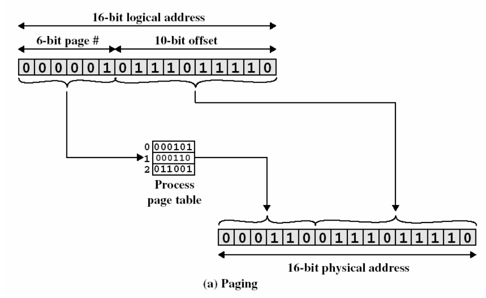
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



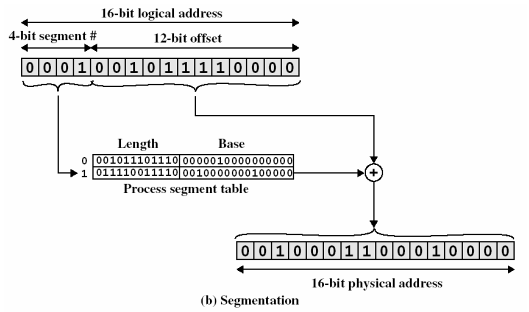
Logical-to-Physical Address Translation

Logical-to-Physical Address Translation (Paging)



(a) Paging

Logical-to-Physical Address Translation (Segmentation)



(b) Segmentation