CSC 553 Operating Systems

Lecture 7 - Memory Management

Memory Management Terms

Frame	**A fixed-length block of main memory.**
Page	**A fixed-length block of data that resides in secondary memory (such as disk). A page of data may temporarily be copied into a frame of main memory.**
Segment	**A variable-length block of data that resides in secondary memory. An entire segment may temporarily be copied into an available region of main memory (segmentation) or the segment may be divided into pages which can be individually copied into main memory (combined segmentation and paging).**

Memory Management Requirements

- Memory management is intended to satisfy the following requirements:
  - Relocation
  - Protection
  - Sharing
  - Logical organization
  - Physical organization

Relocation

- Programmers typically do not know in advance which other programs will be resident in main memory at the time of execution of their program
- Active processes need to be able to be swapped in and out of main memory in order to maximize processor utilization
- Specifying that a process must be placed in the same memory region when it is swapped back in would be limiting
  - May need to *relocate* the process to a different area of memory

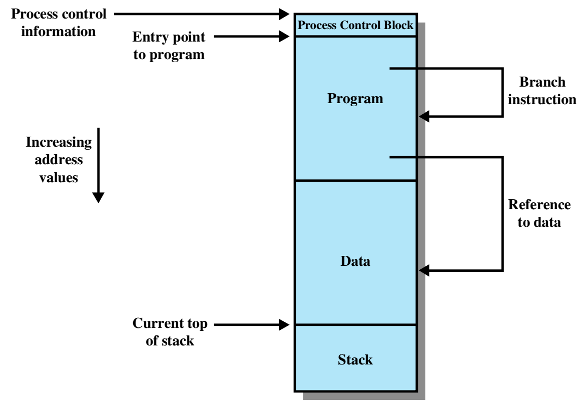


Figure 7.1 Addressing Requirements for a Process

Protection

- Processes need to acquire permission to reference memory locations for reading or writing purposes
- Location of a program in main memory is unpredictable
- Memory references generated by a process must be checked at run time
- Mechanisms that support relocation also support protection

Sharing

- Advantageous to allow each process access to the same copy of the program rather than have their own separate copy
- Memory management must allow controlled access to shared areas of memory without compromising protection
- Mechanisms used to support relocation support sharing capabilities

Logical Organization

- Memory is organized as linear
- Programs are written in modules
  - Modules can be written and compiled independently
  - Different degrees of protection given to modules (read-only, execute-only)
  - Sharing on a module level corresponds to the user’s way of viewing the problem
- Segmentation is the tool that most readily satisfies requirements

Physical Organzation

- Cannot leave the programmer with the responsibility to manage memory
- Memory available for a program plus its data may be insufficient
- Programmer does not know how much space will be available
- ***Overlaying*** allows various modules to be assigned the same region of memory but is time consuming to program

Memory Partitioning

- Memory management brings processes into main memory for execution by the processor
  - Involves virtual memory
  - Based on segmentation and paging
- Partitioning
  - Used in several variations in some now-obsolete operating systems
  - Does not involve virtual memory

Memory Management Techniques

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.
Virtual Memory Paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual Memory Segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

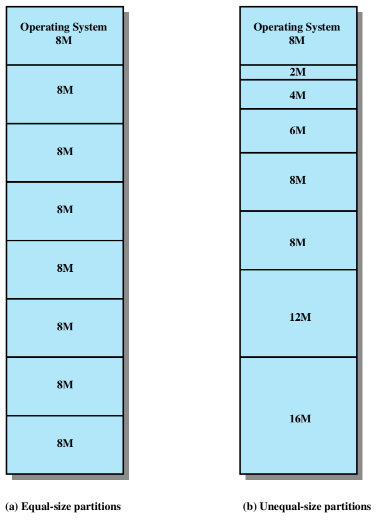


Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

Disadvantages

- A program may be too big to fit in a partition
    - Program needs to be designed with the use of overlays
- Main memory utilization is inefficient
    - Any program, regardless of size, occupies an entire partition
    - ***Internal fragmentation***
        - Wasted space due to the block of data loaded being smaller than the partition

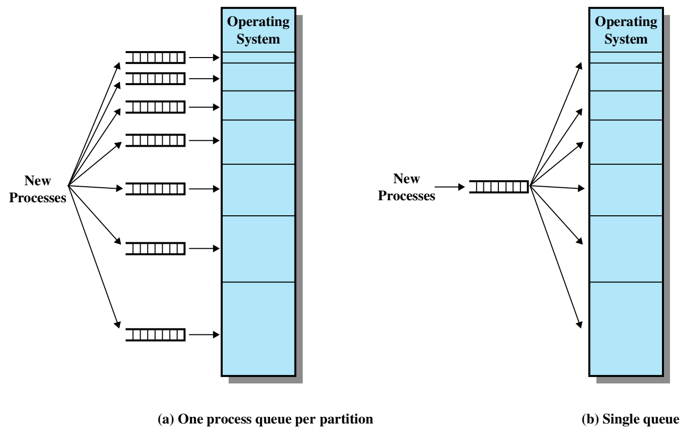


Figure 7.3 Memory Assignment for Fixed Partitioning

Disadvantages

- The number of partitions specified at system generation time limits the number of active processes in the system
- Small jobs will not utilize partition space efficiently

Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as it requires
- This technique was used by IBM's mainframe operating system, OS/MVT

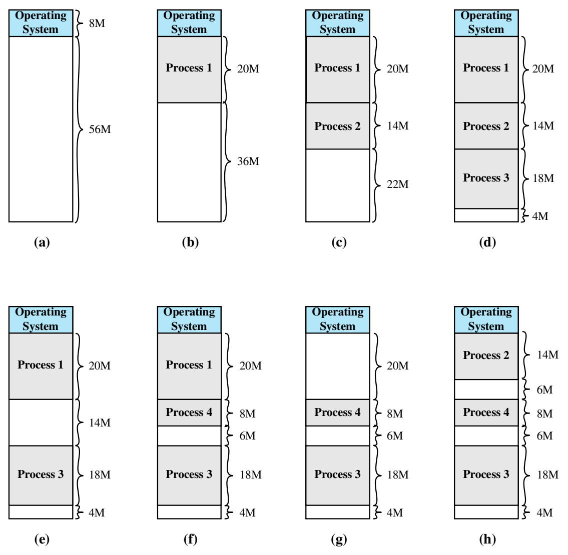


Figure 7.4 The Effect of Dynamic Partitioning

Dynamic Partitioning

- External Fragmentation
  - Memory becomes more and more fragmented
  - Memory utilization declines
- Compaction
  - Technique for overcoming external fragmentation
  - OS shifts processes so that they are contiguous
  - Free memory is together in one block
  - Time consuming and wastes CPU time

Placement Algorithms

- **<u>Best Fit</u>** - Chooses the block that is closest in size to the request
- **<u>First Fit</u>** - Begins to scan memory from the beginning and chooses the first available block that is large enough
- **<u>Next Fit</u>** - Begins to scan memory from the location of the last placement and chooses the next available block that is large enough

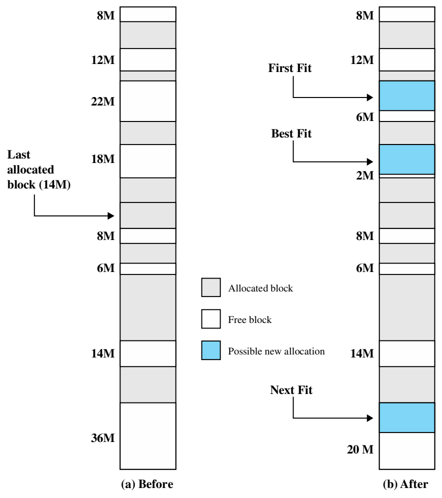


Figure 7.5 Example Memory Configuration before and after Allocation of 16-Mbyte Block

Buddy System

- Comprised of fixed and dynamic partitioning schemes
- Space available for allocation is treated as a single block
- Memory blocks are available of size 2^K *words,* $L \leq K \leq U$, where
  - 2^L = smallest size block that is allocated
  - 2^U = largest size block that is allocated; generally 2^U is the size of the entire memory available for allocation

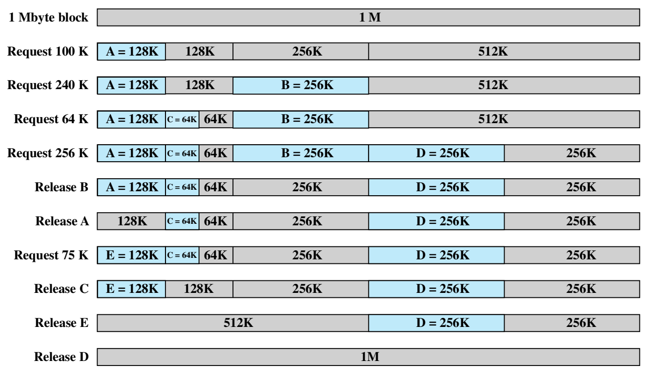


Figure 7.6 Example of Buddy System

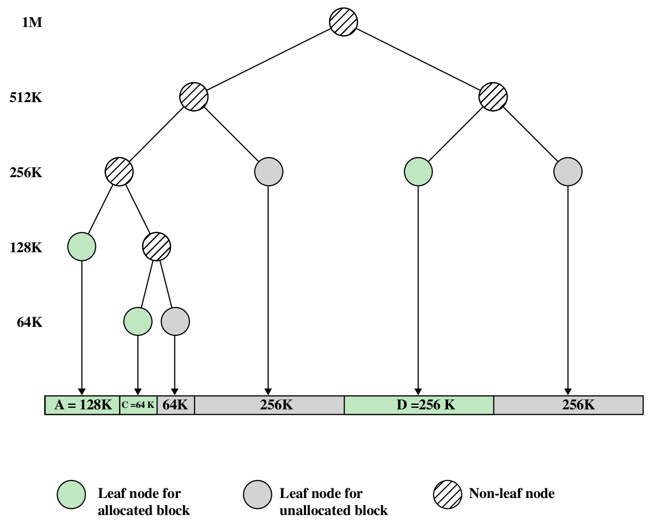


Figure 7.7 Tree Representation of Buddy System

Relocation

- When the fixed partition scheme is used, we can expect a process will always be assigned to the same partition
  - Whichever partition is selected when a new process is loaded will always be used to swap that process back into memory after it has been swapped out
  - In that case, a simple relocating loader can be used
  - When the process is first loaded, all relative memory references in the code are replaced by absolute main memory addresses, determined by the base address of the loaded process

Relocation

- In the case of equal-size partitions and in the case of a single process queue for unequal-size partitions, a process may occupy different partitions during the course of its life
  - When a process image is first created, it is loaded into some partition in main memory; Later, the process may be swapped out
  - When it is subsequently swapped back in, it may be assigned to a different partition than the last time
  - The same is true for dynamic partitioning

Relocation

- When compaction is used, processes are shifted while they are in main memory
  – Thus, the locations referenced by a process are not fixed
  – They will change each time a process is swapped in or shifted

Addresses

- Logical
  - Reference to a memory location independent of the current assignment of data to memory
- Relative
  - A particular example of logical address, in which the address is expressed as a location relative to some known point
- Physical or Absolute
  - Actual location in main memory

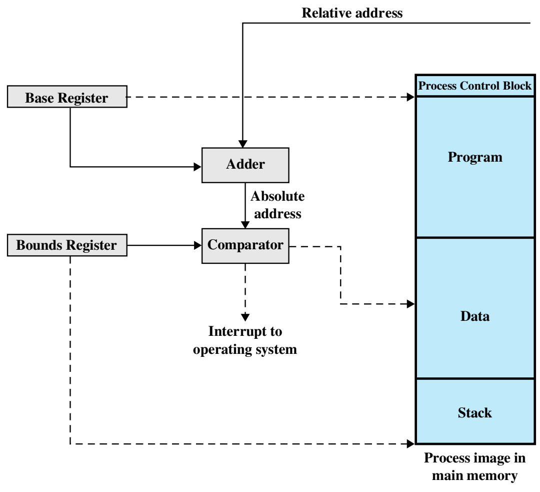


Figure 7.8 Hardware Support for Relocation

Paging

- Partition memory into equal fixed-size chunks that are relatively small
- Process is also divided into small fixed-size chunks of the same size
- **Pages** - Chunks of a process
- **Frames** - Available chunks of memory

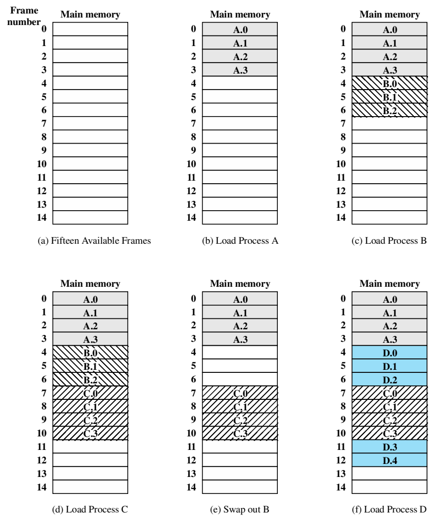


Figure 7.9 Assignment of Process Pages to Free Frames

Page Table

- Maintained by operating system for each process
- Contains the frame location for each page in the process
- Processor must know how to access for the current process
- Used by processor to produce a physical address

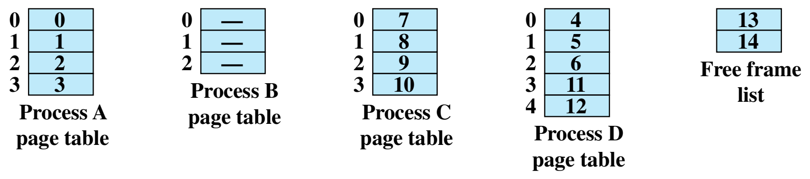


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

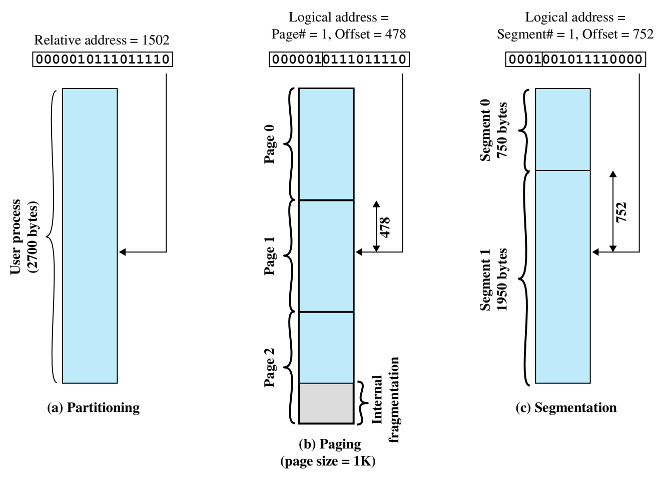
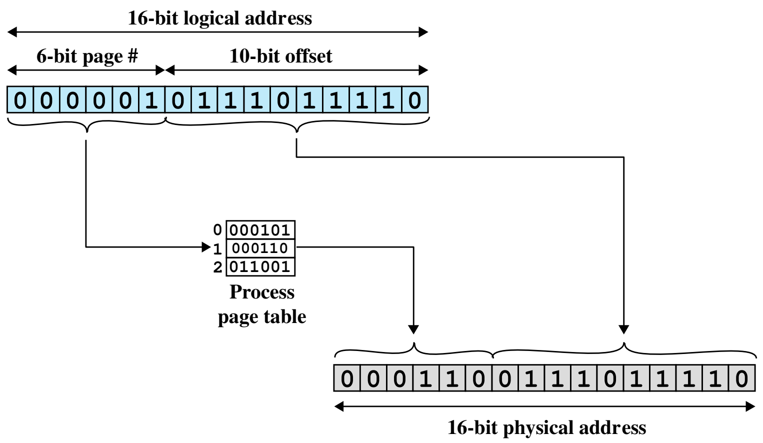


Figure 7.11 Logical Addresses



(a) Paging

Segmentation

- A program can be subdivided into segments
  - May vary in length
  - There is a maximum length
- Addressing consists of two parts:
  - Segment number
  - An offset
- Similar to dynamic partitioning
- Eliminates internal fragmentation

Segmentation

- Usually visible
- Provided as a convenience for organizing programs and data
- Typically the programmer will assign programs and data to different segments

Segmentation

- For purposes of modular programming the program or data may be further broken down into multiple segments
  - The principal inconvenience of this service is that the programmer must be aware of the maximum segment size limitation

Address Translation

- Another consequence of unequal size segments is that there is no simple relationship between logical addresses and physical addresses

Necessary Steps for Address Translation

1. Extract the segment number as the leftmost *n* bits of the logical address
2. Use the segment number as an index into the process segment table to find the starting physical address of the segment
3. Compare the offset, expressed in the rightmost *m* bits, to the length of the segment. If the offset is greater than or equal to the length, the address is invalid
4. The desired physical address is the sum of the starting physical address of the segment plus the offset

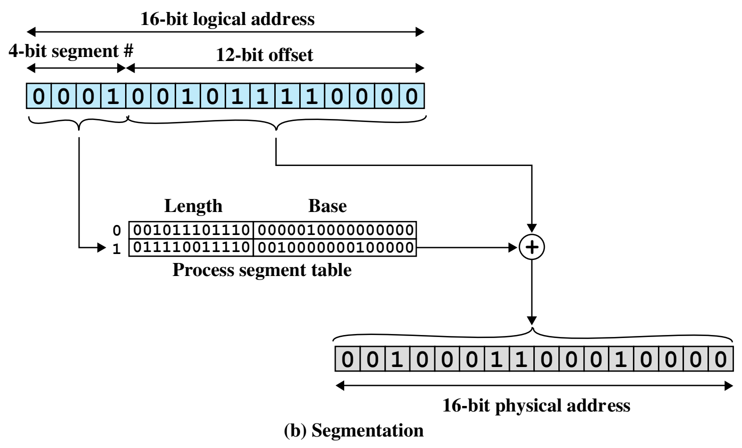


Figure 7.12 Examples of Logical-to-Physical Address Translation

Summary

- Memory management requirements
    - Relocation
    - Protection
    - Sharing
    - Logical organization
    - Physical organization

Summary

- Memory partitioning
    - Fixed partitioning
    - Dynamic partitioning
    - Buddy system
    - Relocation

Summary

- Paging
- Segmentation