CS415 Module 7 Part A - Memory Management

Athens State University

Outline

Contents

1 Memory

What Happens When You Start A Process?

- 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

Why Addressing Is Hard...

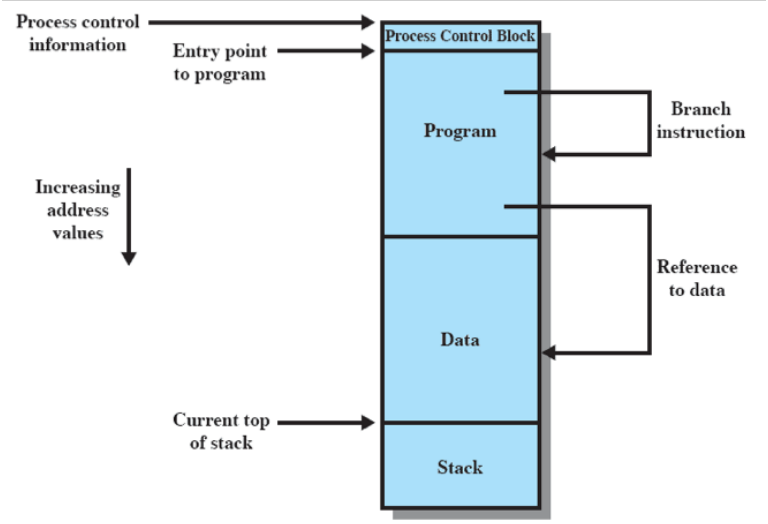


Figure 7.1 Addressing Requirements for a Process

- Must deal with the problems of relocation
  - Address in programs have to be relative
  - Operating system must work with hardware to convert relative addresses to physical addresses

Processes have to be protected

- Processes need to acquire permission to reference memory locations for reading and 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 must also support protection

Logical Organization of Memory

- Memory is organized as a linear array

Programs are written in modules

  - Modules can be written and compile independently
  - Different degrees of protection can be given to modules
  - Shared on a module level corresponds to the user's way of viewing the problem

- Segmentation is the tool that most readily satisfies these requirements

Physical Organization of Memory

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

2 Virtual Memory

Virtual Memory

- **Virtual memory**: separation of user logical memory from physical memory
  - Only part of the program needs be in memory for execution
  - Logical address space can be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation
  - More programs running concurrently
  - Less I/O needed to load or swap processes
- Virtual memory can be implemented via: Demand paging or Demand segmentation

Three Key Terms

Frame A fixed-length block of main memory

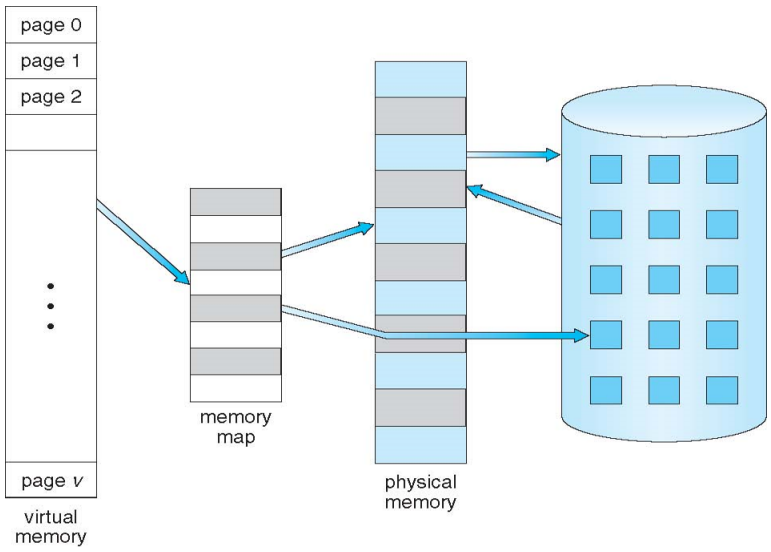
Page A fixed-length block of data residing in secondary memory (such as a disk).

- A page of data my be copied into a frame of main memory

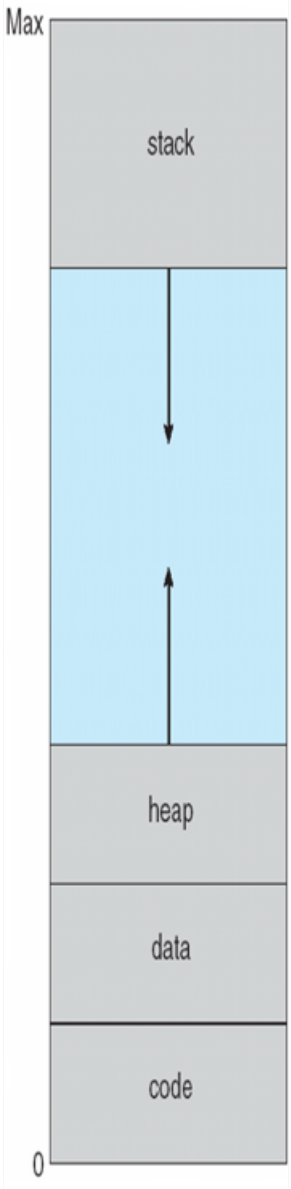
Segment A variable-length block of data that resides in secondary memory

- An entire segment may be copied into an available region of main memory
- Segment may divided into pages that can be individually copied into main memory

Virtual Memory: Much Larger Than Physical Memory



Virtual Address Space



- Enables sparse address spaces with holes left for growth, dynamically linked libraries, etc
- System libraries shared via mapping into virtual address space
- Shared memory by mapping pages read-write into virtual address space
- Pages can be shared during fork(), speeding process creation

Address Types

Logical

- Reference to a memory location independent of the current assignment of data to memory

Relative

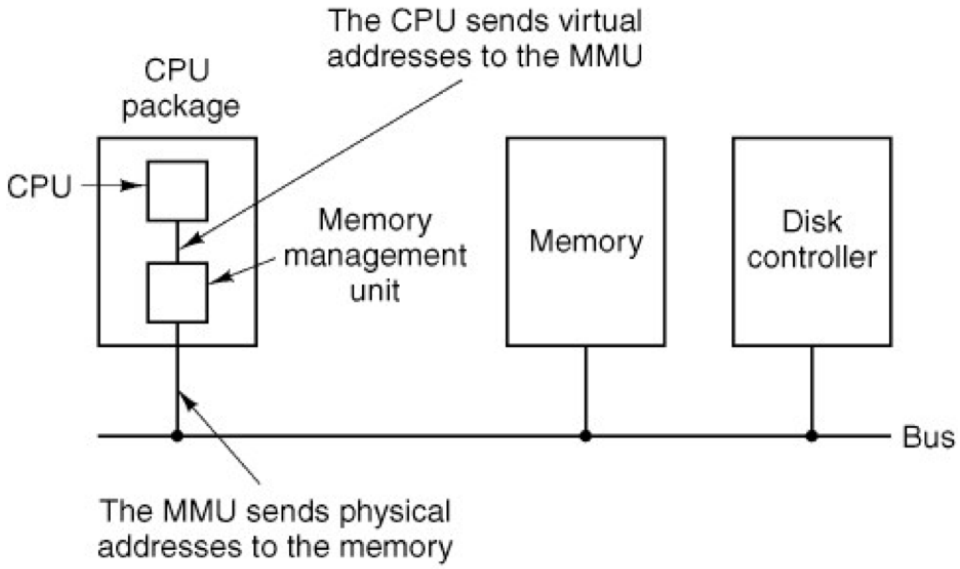
- Address is expressed as a location relative to some known point

Physical or Absolute

- Actual location in main memory

3 Hardware Support

Memory Management Units



Hardware Support for Relocation

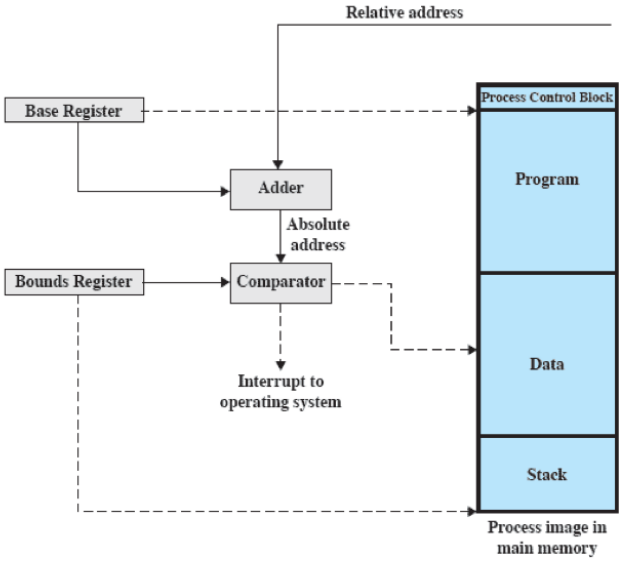


Figure 7.8 Hardware Support for Relocation

- Registers keep track of the "base" and "bound" locations of the process
- Two step process
  - Relative address is added to the base register
  - Result compared against the bounds register

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

Demand Paging

- Two options
  - Bring entire process into memory at load time
  - Bring a page into memory only when it is needed
    - Less I/O needed, no unnecessary I/O

    * Less memory needed
    * Faster response
    * More users

- Page is needed generates a reference to that page
  - Invalid reference, abort access
  - Not-in-memory, load page into frame

Demand Paging

- *Lazy swapper*: never swap a page into memory unless page is needed
- OS keeps track of free frames
- On process load, pages for a process are loaded into available frames

Demand Paging

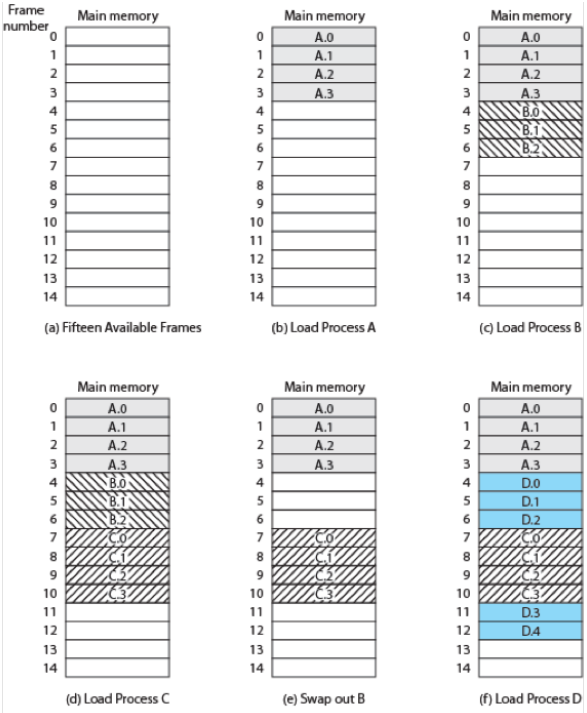
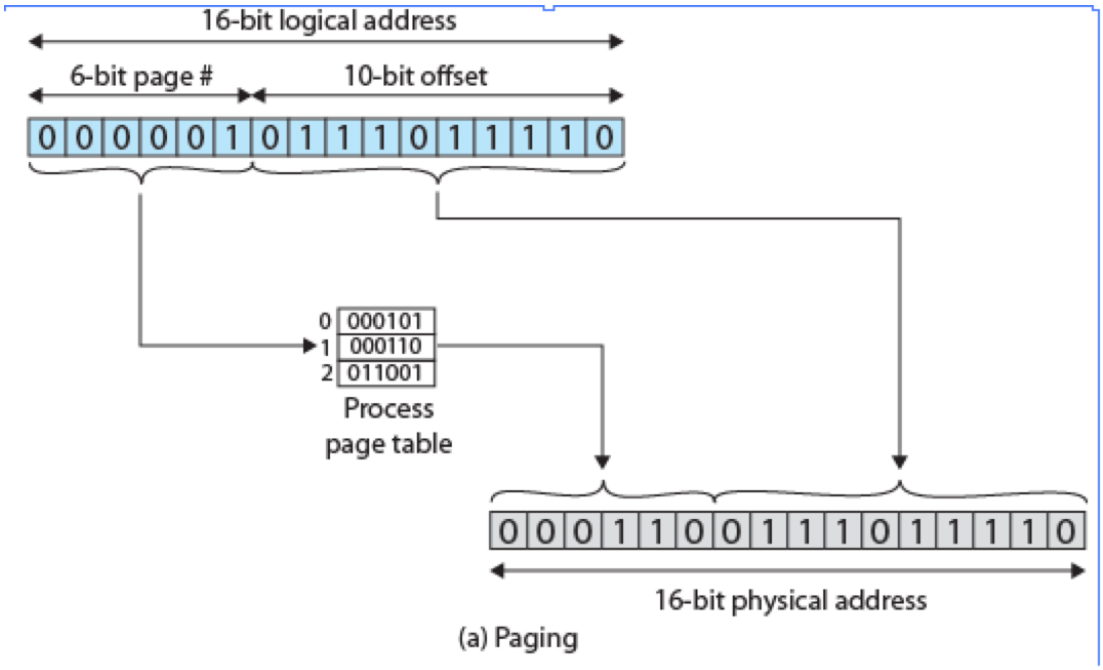


Figure 7.9 Assignment of Process Pages to Free Frames

Page Table

- Need to keep track of the frame locations for each page of process
- Logical address becomes pair of page number and offset
- Physical address becomes pair of frame number and offset
- Address resolution is an interface issue between the OS and the CPU's MMU

Logical to Physical Address



In our example, we have the logical address 0000010111011110, which is page number 1, offset 478. Suppose that this page is residing in main memory frame 6 binary 000110. Then the physical address is frame number 6, offset 478: 0001100111011110.

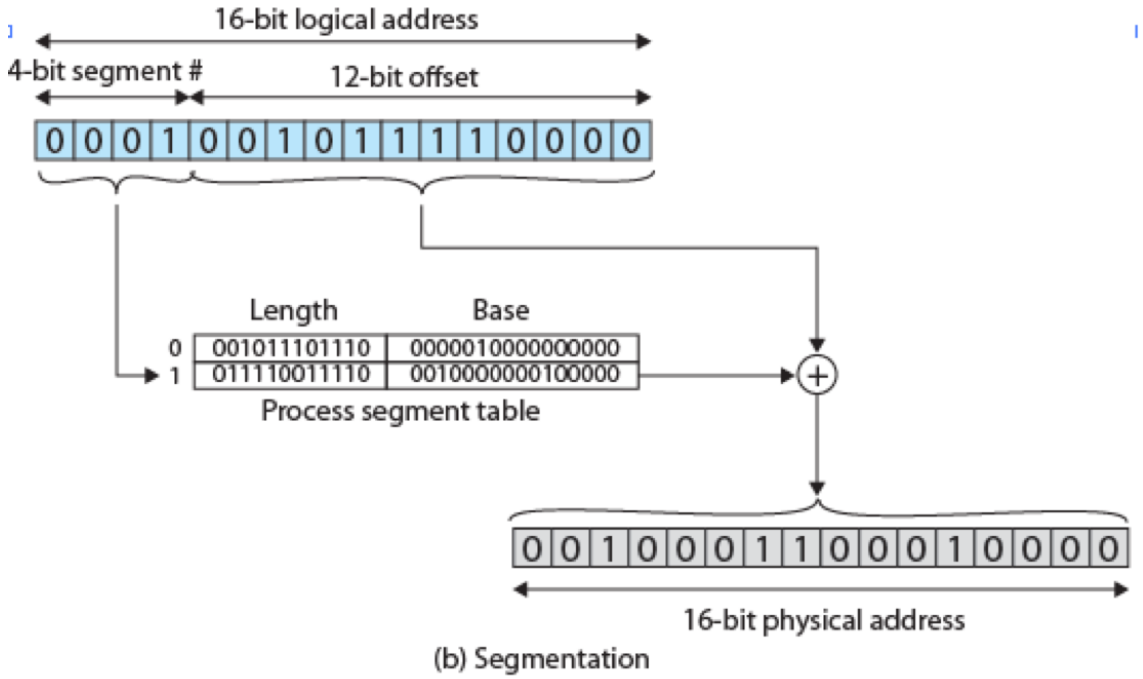
Segmentation

- Alternate to paging where a program is subdivided in variable-length segments
- Addressing consists of two parts: segment number and offset
- Eliminates internal fragmentation of memory

A user program can be subdivided using segmentation, in which the program and its associated data are divided into a number of segments . It is not required that all segments of all programs be of the same length, although there is a maximum segment length. As with paging, a logical address using segmentation consists of two parts, in this case a segment number and an offset.

Because of the use of unequal-size segments, segmentation is similar to dynamic partitioning. In the absence of an overlay scheme or the use of virtual memory, it would be required that all of a program's segments be loaded into memory for execution. The difference, compared to dynamic partitioning, is that with segmentation a program may occupy more than one partition, and these partitions need not be contiguous. Segmentation eliminates internal fragmentation but, like dynamic partitioning, it suffers from external fragmentation. However, because a process is broken up into a number of smaller pieces, the external fragmentation should be less.

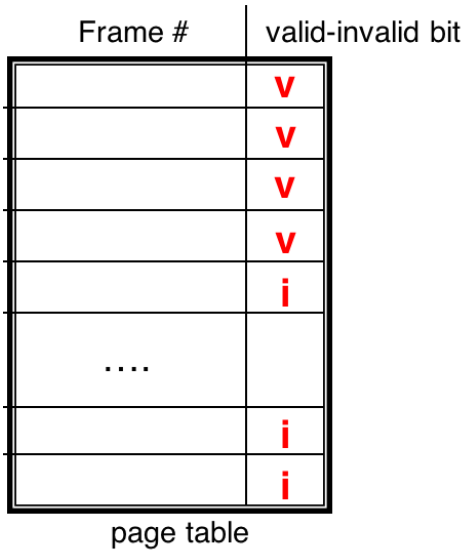
The Valid-Invalid Bit



(b) Segmentation

In our example, we have the logical address 0001001011110000, which is segment number 1, offset 752. Suppose that this segment is residing in main memory starting at physical address 0010000000100000. Then the physical address is 0010000000100000 + 001011110000 0010001100010000.

Logical to Physical Address



- Each page table entry contains a *valid-invalid* bit
  - If bit is set (valid), then page is memory resident
- Initially this bit is set to invalid for all pages
- During address translation, if this bit is not set then we have an instance of a **page fault**

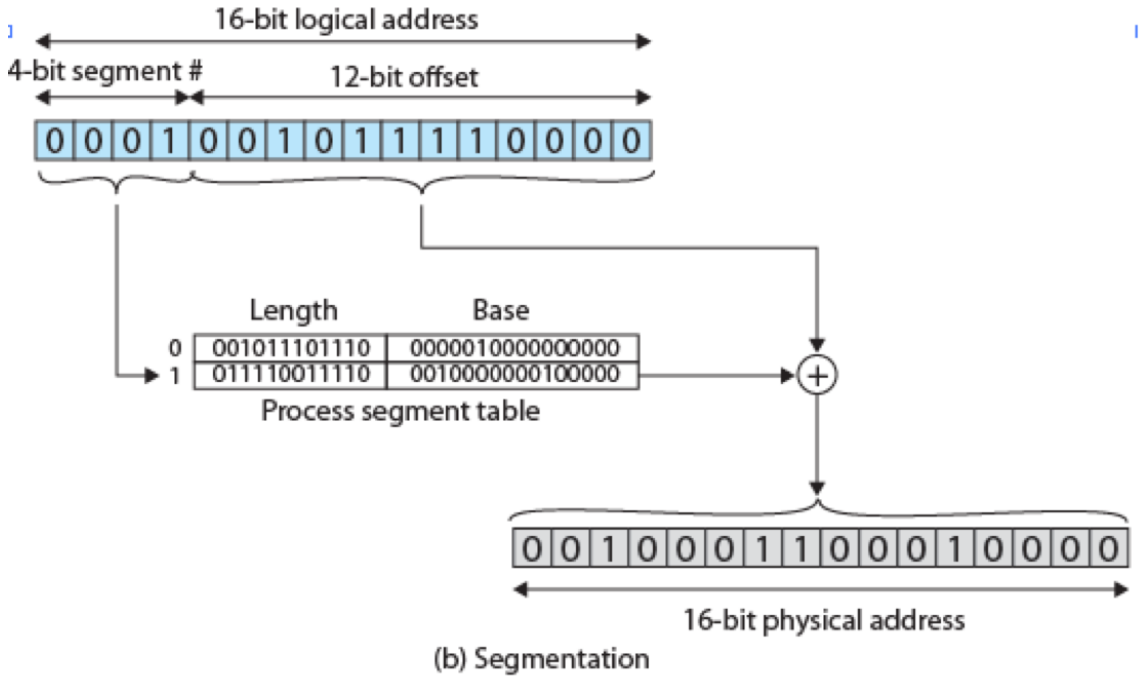
Page Fault

- A page fault occurs on first reference to a page
- Operating system determines if valid reference and page not in frame
- Get an empty frame
- Swap page into frame via usual disk operation
- Reset tables to indicate page in memory
- Restart the instruction that caused the page fault

Page Faults and Demand Paging

- Extreme case - when process starts, it has *no* pages in memory
  - OS sets instruction pointer to first instruction of process, which is not in memory resulting in page fault
  - And for every other process pages on first access
  - Pure demand paging
- Potential of multiple page faults, impact lessened due to the *principle of locality of reference*

Steps in Handling A Page Fault



(b) Segmentation

4 Key Points

Key Points

- Types of scheduling
- Scheduling of criteria
- Non-preemptive vs. preemptive