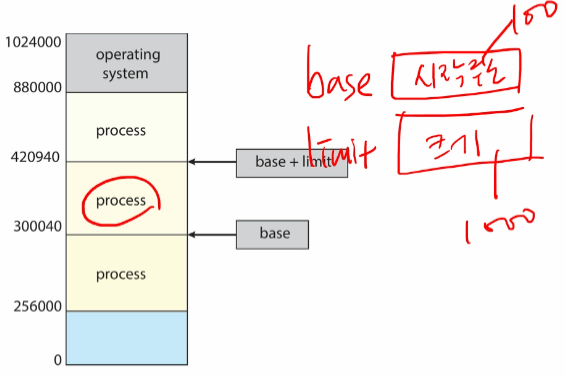
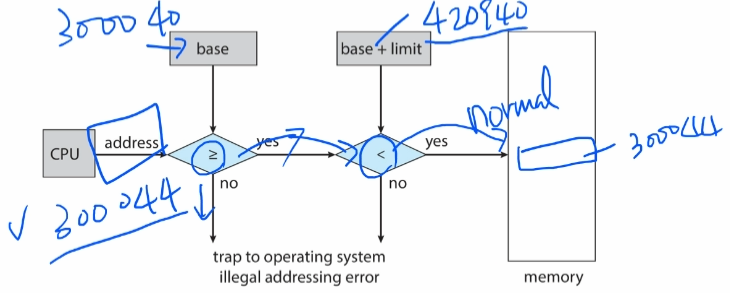
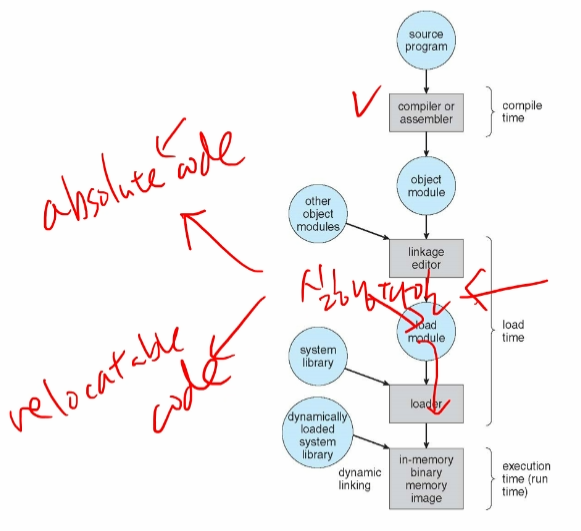
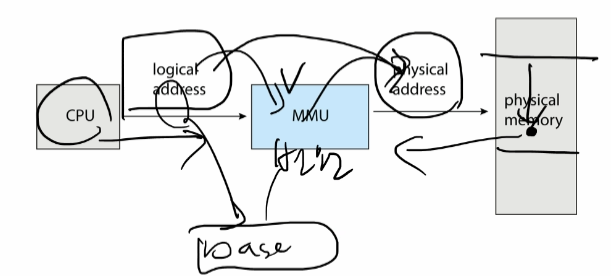
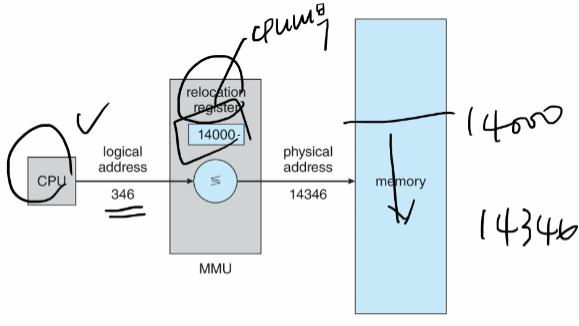
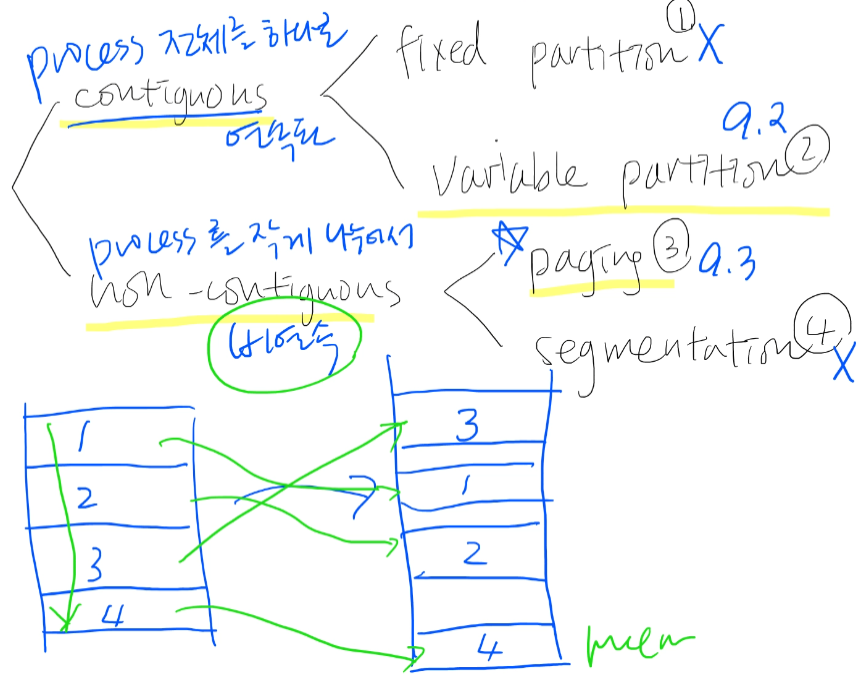
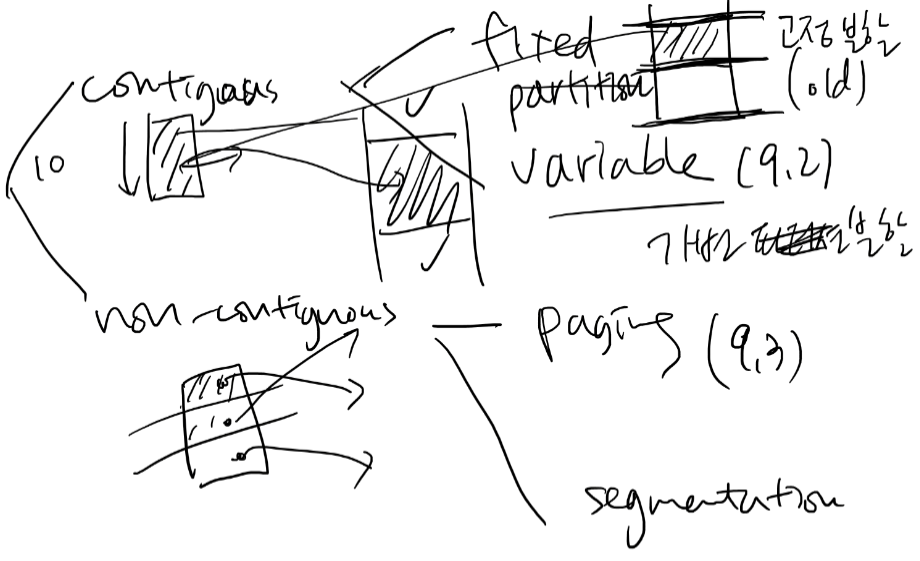
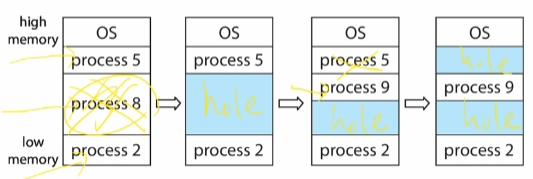
Chapter 9: Main Memory; 메인 메모리

1. Background; 배경
   * 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  
     메인 메모리와 레지스터들은 CPU가 직접 접근할 수 있는 유일한 범용 저장장치이다.
   * Memory unit only sees a stream of:
     + addresses + read requests, or
     + address + data and write requests
   * Register access is done in one CPU clock (or less)  
     각 CPU 코어에 내장된 레지스터들은 일반적으로 CPU 클록(clock)의 1사이클(cycle) 내에 접근이 가능하다
   * Main memory can take many cycles, causing a stall  
     메인 메모리의 접근을 완료하기 위해서는 많은 CPU 클록 틱 사이클이 소요되며, 이 경우 CPU가 필요한 데이터가 없어서 명령어를 수행하지 못하고 지연되는(stall) 현상이 발생하게 된다
   * Cache sits between main memory and CPU registers
   * Protection of memory required to ensuer correct operation
   1. 기본 하드웨어
      1. Protection
         1. Need to ensure that a process can access only access those addresses in it address space
         2. We can provide this protection by using a pair of base and limit registers define the logical address space of a process  
            
      2. Hadrware Address Protection
         1. CPU must check every memory access generated in user mode to be sure it is between base and limit for that user  
            
         2. the instructions to loading the base and limit registers are privileged
   2. Address Binding; 주소의 할당
      1. Programs on disk, ready to be brought into memory to execute, form an input queue
         1. Without support, must be loaded into address 0000
      2. Addresses represented in different ways at different stages of a program’s life
         1. Source code addresses usually symbolic
         2. Compiled code addresses bind to relocatable addresses
            1. i.e. “14 bytes from beginning of this module”
         3. Linker or loader will bind relocatable addresses to absolute addresses
            1. i.e. 74014
         4. Each binding maps one address space to another
      3. Binding of Instuctions and Data to Memory
         1. Address binding of instructions and data to memory addresses can happen at three different stages
            1. Compile time: If memory location known a priori, absolute code can be generated; must recompile code if starting location changes
            2. Load time: Must generate relocatable code if memory location is not known at compile time
            3. Execution time: Binding delayed until run time if the process can be moved during its execution from ome memory segment to another

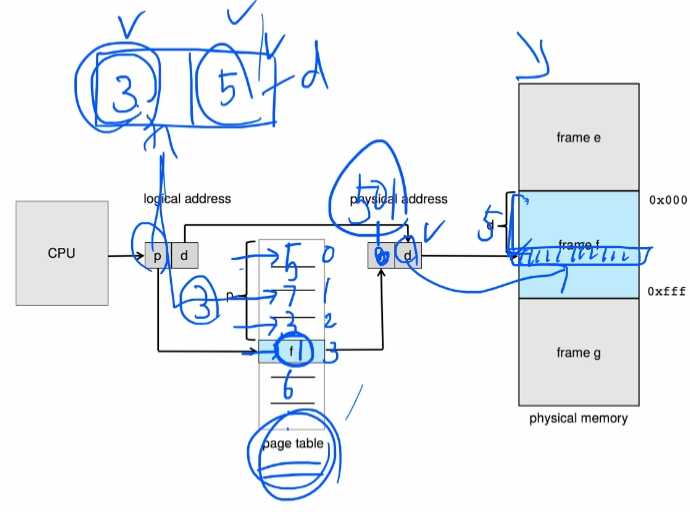
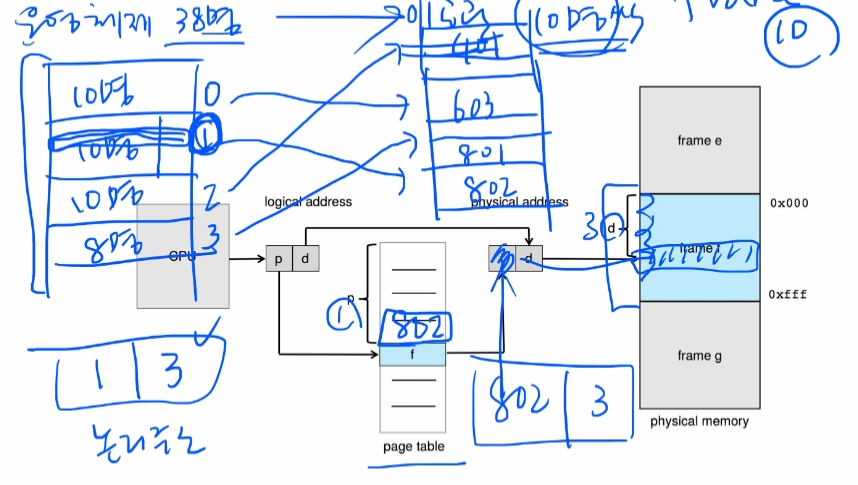
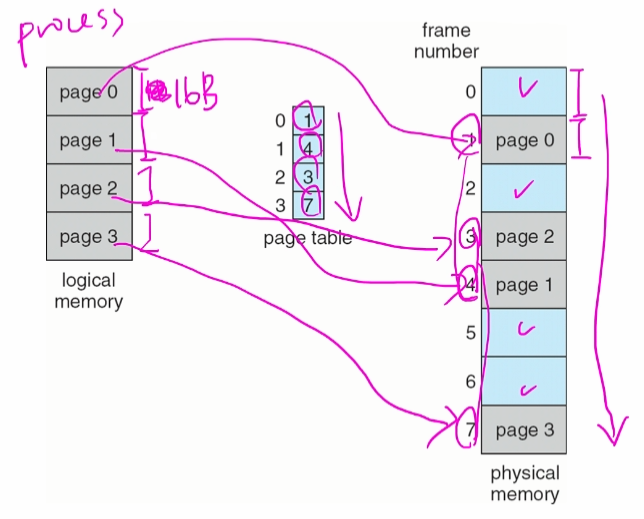
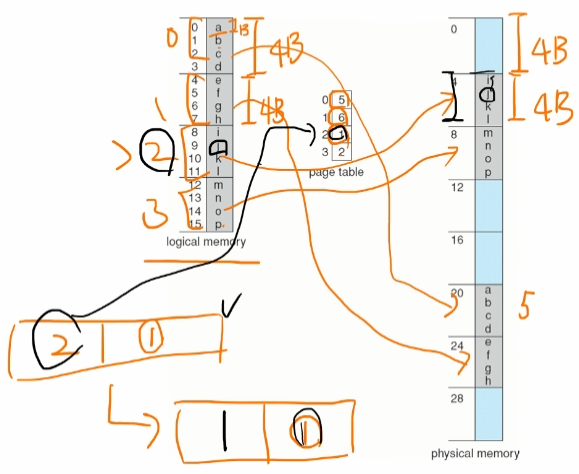
Need hardware support for address maps (e.g., base and limit registers)

* + 1. Multistep Processing of a User Program  
       
  1. Logical-Versus Physical-Address Space; 논리 대 물리 주소 공간
     1. The concept of a logical address space that is bound to a separate physical address space is central to proper memory management
        1. Logical address – generated by the CPU; also referred to as virtual address
        2. Physical address – address seen by the memory unit
     2. 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
     3. Logical address space is the set of all logical addresses generated by a program
     4. Physical address space is the set of all physical addresses generated by a program
     5. Memory-Management Unit (MMU)
        1. Hardware device that at run time maps virtual to physical address  
           
        2. Many methods possible, covered in the rest of this chapter
        3. Consider simple scheme. which is a generalization of the base-register scheme.
        4. The base register now called relocation register
        5. The value in the relocation register is added to every address generated by a user process at the time it is sent to memory  
           

1. Contiguous Memory Allocation; 연속 메모리 할당  
     
   
   * Main memory must support both OS and user processes
   * Limited resource, must allocate efficiently
   * Contiguous allocation is one early method
   * Main memory usually into two partitions:
     + Resident operating system, usually held in low memory with interrupt vector
     + User processes then held in high memory
     + Each process contained in single contiguous section of memory
   1. Memory Allocation; 메모리 할당
      1. Variable Partition; 가변 분할
         1. Multiple-partition alocation
            1. Degree of multiprogramming limited by number of partitions
            2. Variable-partition sizes for efficiency (sized to a given process’ needs)
            3. Hole – block of available memory; holes of various size are scattered throughout memory
            4. When a process arrives, it is allocated memory from a hole large enough to accommodate it
            5. Process exiting frees its partition, adjacent free partitions combined
            6. Operating system maintains information about:  
               a) allocated partitions b) free partitions (hole)  
               
      2. Dynamic storage-Allocation Problem  
         How to satisfy a request of size n from a list of free holes?
         1. First-fit: Allocate the first hole that is big enough
         2. Best-fit: Allocate the smallest hole that is big enough; must search entire list, unless ordered by size
            1. Produces the smallest leftover hole
         3. Worst-fit: Allocate the largest hloe; must also search entire list
            1. Produces the largest leftover hole

First-fit and best-fit better than worst-fit in terms of speed and storage utilization

* 1. Fragmentation; 단편화
     1. External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous
     2. Internal Fragmentaiton – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
     3. First fit analysis reveals that given N blocks allocated, 0.5 N blocks lost to fragmentation -> 50-percent rule
     4. Reduce external fragmentation by compaction

1. Paging; 페이징
   1. Basic Method; 기본 방법
      1. Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the letter is available
         1. Avoids external fragmentation
         2. Avoids problem of varying sized memory chunks
      2. Divide physical memory into fixed-sized blocks called frames
         1. Sized is power of 2, between 512 bytes and 16Mbytes
      3. Divide logical memory into blocks of same size called pages
      4. Keep track of all free frames
      5. To run a program of size N pages, need to find N free frames and load program
      6. Set up a page table to translate logical to physical addresses
      7. Still have internal fragmentation
      8. Address Translation Scheme
         1. Address generated by CPU is divided into:
            1. Page number (p) – used as an index into a page table which contains base address of each page in physical memory
            2. Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit
            3. For given logical address space 2^m and page size 2^n
      9. Paging Hardware  
           
         
      10. Paging Model of Logical and Physical Memory  
          
      11. Paging Example  
          Logical address: n = 2 and m = 4. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages)  
          
      12. Paging – Calculating internal fragmentation
          1. Page size = 2,048 bytes
          2. Process size = 72,766 bytes
          3. 35 pages + 1,086 bytes
          4. Internal fragmentation of 2,048 – 1,086 = 962 bytes
          5. So small frame sizes desirable?
          6. But each page table entry takes memory to track
      13. Paging Hardware With TLB  
          