CSC 4320/6320: Operating Systems

Georgia State University

Chapter 09: Main Memory

Spring 2025

Instructor: Dr. Md Mahfuzur Rahman
Department of Computer Science, GSU

Disclaimer

The slides to be used in this course have been created, modified, and adapted from multiple sources:

- *The slides are copyright Silberschatz, Galvin and Gagne, 2018. The slides are authorized for personal use, and for use in conjunction with a course for which Operating System Concepts is the prescribed text. Instructors are free to modify the slides to their taste, as long as the modified slides acknowledge the source and the fact that they have been modified.*

Chapter 9: Memory Management

Georgia State University

- Background
- Contiguous Memory Allocation
- Paging
- Structure of the Page Table
- Swapping
- Example: The Intel 32 and 64-bit Architectures
- Example: ARMv8 Architecture

Objectives

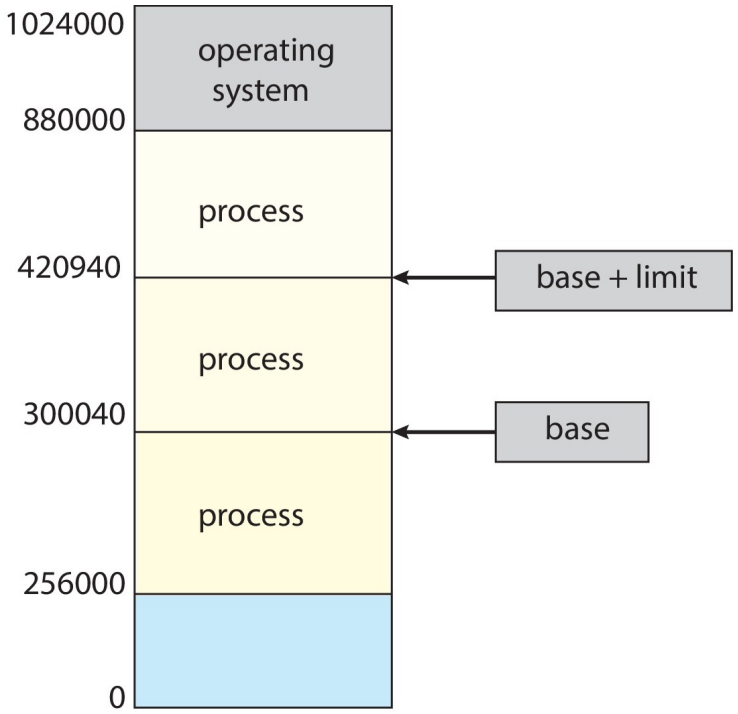
- To provide a detailed description of various ways of organizing memory hardware
- To discuss various memory-management techniques,
- To provide a detailed description of the Intel Pentium, which supports both pure segmentation and segmentation with paging

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
- 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)
- Main memory can take many cycles, causing a **stall (CPU is waiting for the data)**
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation (usually managed by main memory hardware, not by OS)

Protection

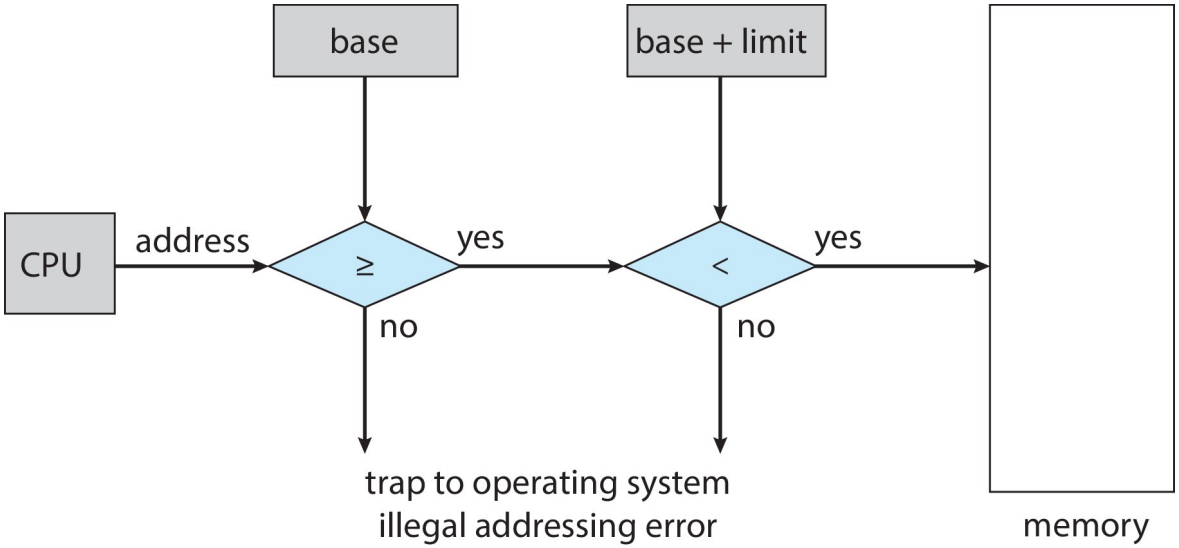
- Need to ensure that a process can access only those addresses in its address space.
- We can provide this protection by using a pair of **base** and **limit registers** define the logical address space of a process



For example, if the base register holds 300040 and the limit register is 120900, then the program can legally access all addresses from 300040 through 420939 (inclusive).

Hardware Address Protection

- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user



- the instructions to loading the base and limit registers are privileged and only by OS.

Address Binding

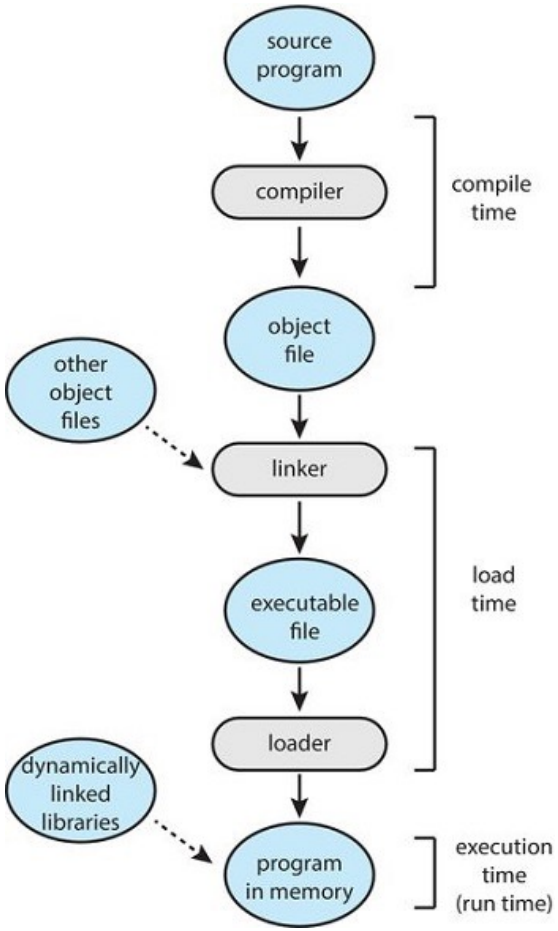
- Programs on disk, ready to be brought into memory to execute form an **input queue**
  - Without support, must be loaded into address 0000
- Most systems allow a user process to reside in any part of the physical memory.
- The address space of the computer may start at 00000, the first address of the user process need not be 00000.
- Addresses represented in different ways at different stages of a program's life
  - Source code addresses usually symbolic (such as the variable count)
  - Compiled code addresses **bind** to relocatable addresses
    - i.e., "14 bytes from beginning of this module"
  - Linker or loader will bind relocatable addresses to absolute addresses
    - i.e., 74014
  - Each binding maps one address space to another

Binding of Instructions and Data to Memory

Georgia State University

- 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
  - **Load time**: Must generate **relocatable code** if memory location is not known at compile time. In this case, final binding is delayed until load time.
  - **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)

Multistep Processing of a User Program



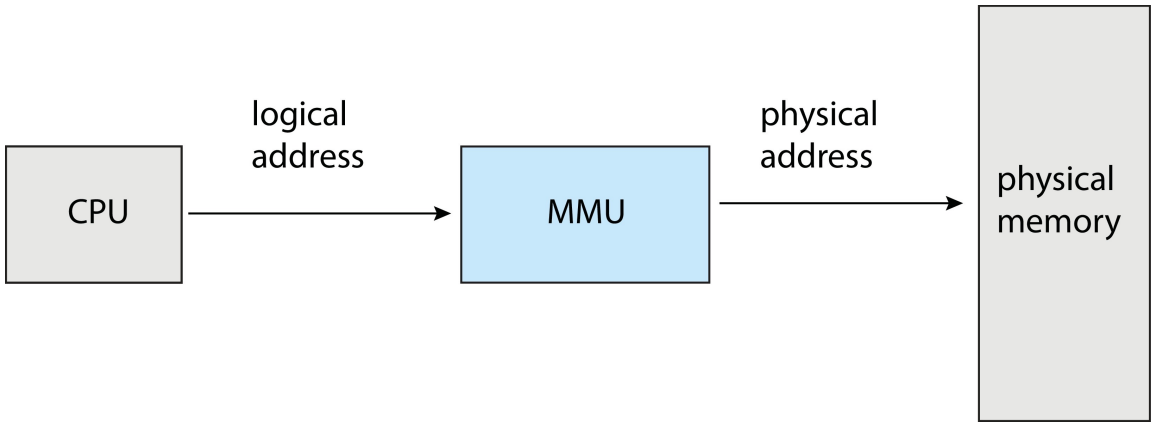
Logical vs. Physical Address Space

Georgia State University

- 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 and load-time address-binding schemes (static binding); logical (virtual) and physical addresses differ in execution-time address-binding scheme
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program

Memory-Management Unit (MMU)

- Hardware device that at run time maps virtual/logical to physical address



- Many methods possible, covered in the rest of this chapter

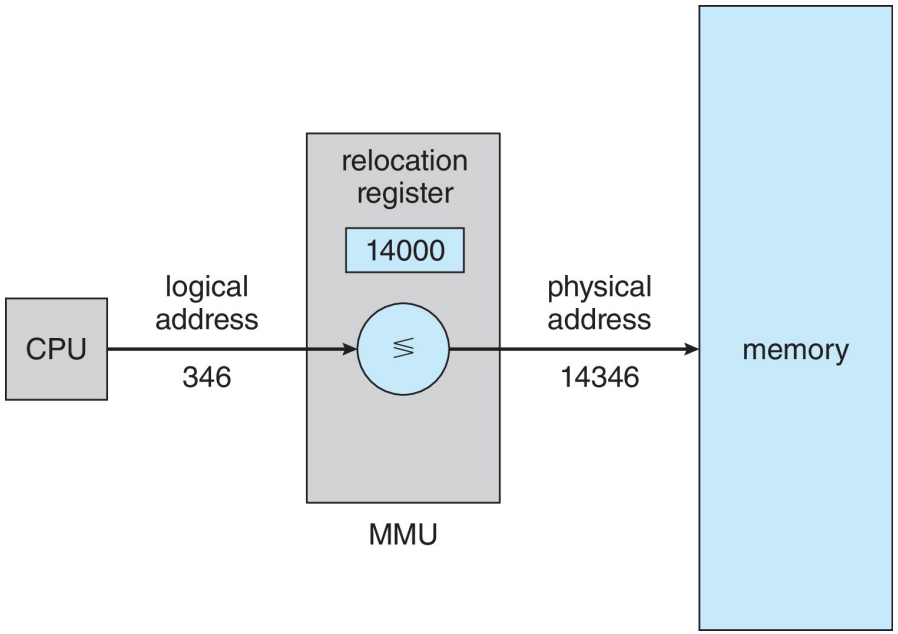
Memory-Management Unit (Cont.)

Georgia State University

- Consider simple scheme. which is a generalization of the base-register scheme.
- The base register now called **relocation register**
- 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
  - Execution-time binding occurs when reference is made to location in memory
  - Logical address bound to physical addresses

Memory-Management Unit (Cont.)

- Consider simple scheme. which is a generalization of the base-register scheme.
- The base register now called **relocation register**
- The value in the relocation register is added to every address generated by a user process at the time it is sent to memory



Dynamic Loading

- The entire program does need to be in memory to execute
- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- Useful when large amounts of code are needed to handle infrequently occurring cases (such as error routines)
- No special support from the operating system is required
  - Implemented through program design
  - OS can help by providing libraries to implement dynamic loading

Dynamically Linked Libraries

- **Static linking** – system libraries and program code combined by the loader into the binary program image
- **Dynamic linking** –linking postponed until execution time
- Without DL, each program on a system must include a copy of its language library (or the routines referenced by the program) in the executable image.
- Increases the size of an executable image, also may waste main memory.
- DLLs libraries can be shared among multiple processes, so that only one instance of the DLL in main memory.
- When a program references a routine that is in a DL, the loader locates the DLL, loading it into memory if necessary.
- Adjusts addresses that reference functions in the dynamic library to the location in memory where the DLL is stored.

Dynamic Linking

- Unlike dynamic loading, dynamic linking and shared libraries generally require help from the operating system.
- the operating system can/must check to see whether the needed routine is in another process's memory space or can allow multiple processes to access the same memory addresses.
- 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. If needed, it asks the loader to load the routine.
- Dynamic linking is particularly useful for libraries
- System also known as **shared libraries**

Questions

1. Absolute code can be generated for _____.
A) compile-time binding ✓
B) load-time binding
C) execution-time binding
D) interrupt binding

2. In a dynamically linked library, _____.
A) loading is postponed until execution time
B) system language libraries are treated like any other object module
C) more disk space is used than in a statically linked library
D) a stub is included in the image for each library-routine reference ✓

3. The mapping of a logical address to a physical address is done in hardware by the _____.
A) memory-management-unit (MMU) ✓
B) memory address register
C) relocation register
D) dynamic loading register

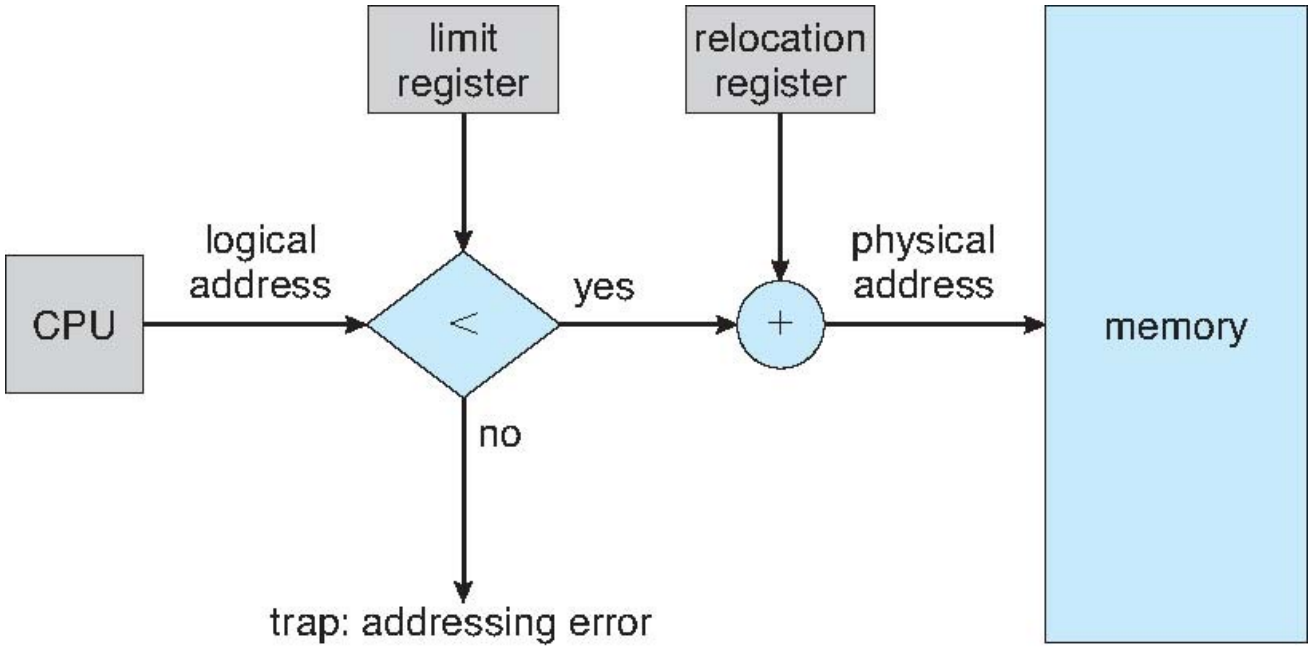
Contiguous Allocation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually divided into two **partitions**:
  - one for the operating system and one for the user processes.
  - Resident operating system, usually held in low memory (addresses) with interrupt vector
  - User processes then held in high memory addresses.
  - Each process contained in single contiguous section of memory

Contiguous Allocation (Cont.)

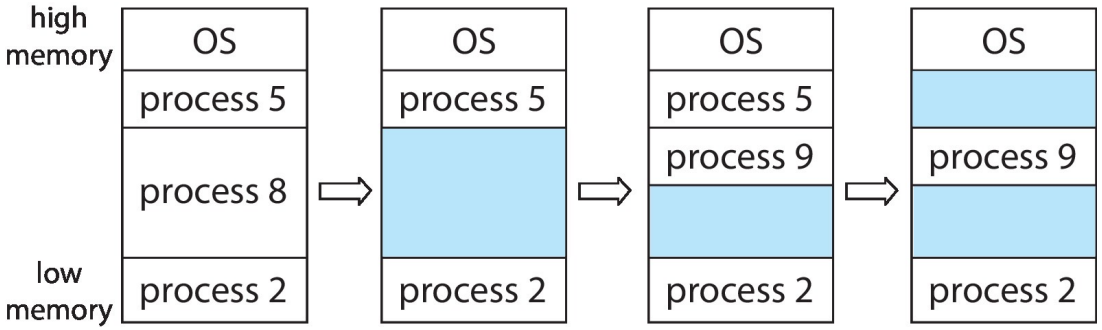
- Memory address protection
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically*
  - Can then allow actions such as kernel code being **transient** and kernel changing size.
    - For example, a device driver can be loaded and removed as needed.

Hardware Support for Relocation and Limit Registers



Variable Partition

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
  - **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
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    a) allocated partitions b) free partitions (hole)



General Scheme of Allocation

- If the hole is too large, it is split into two parts. One part is allocated to the arriving process; the other is returned to the set of holes.
- When a process terminates, it releases its block of memory, which is then placed back in the set of holes.
- If the new hole is adjacent to other holes, these adjacent holes are merged to form one larger hole.

Dynamic Storage-Allocation Problem

Georgia State University

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, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the ***largest*** hole; must also search entire list
  - Produces the largest leftover hole (may be more useful than smaller left over)

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

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
- First fit analysis reveals that given *N* blocks allocated, 0.5 *N* blocks lost to fragmentation
  - 1/3 may be unusable -> **50-percent rule**

Fragmentation (Cont.)

- 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
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers
- Now consider that backing store has same fragmentation problems

Questions

1. _____ is the dynamic storage-allocation algorithm which results in the smallest leftover hole in memory.
A) First fit
B) Best fit ✓
C) Worst fit
D) None of the above

2. _____ is the dynamic storage-allocation algorithm which results in the largest leftover hole in memory.
A) First fit
B) Best fit
C) Worst fit ✓
D) None of the above

3. Which of the following is true of compaction?
A) It can be done at assembly, load, or execution time.
B) It is used to solve the problem of internal fragmentation.
C) It cannot shuffle memory contents.
D) It is possible only if relocation is dynamic and done at execution time. ✓