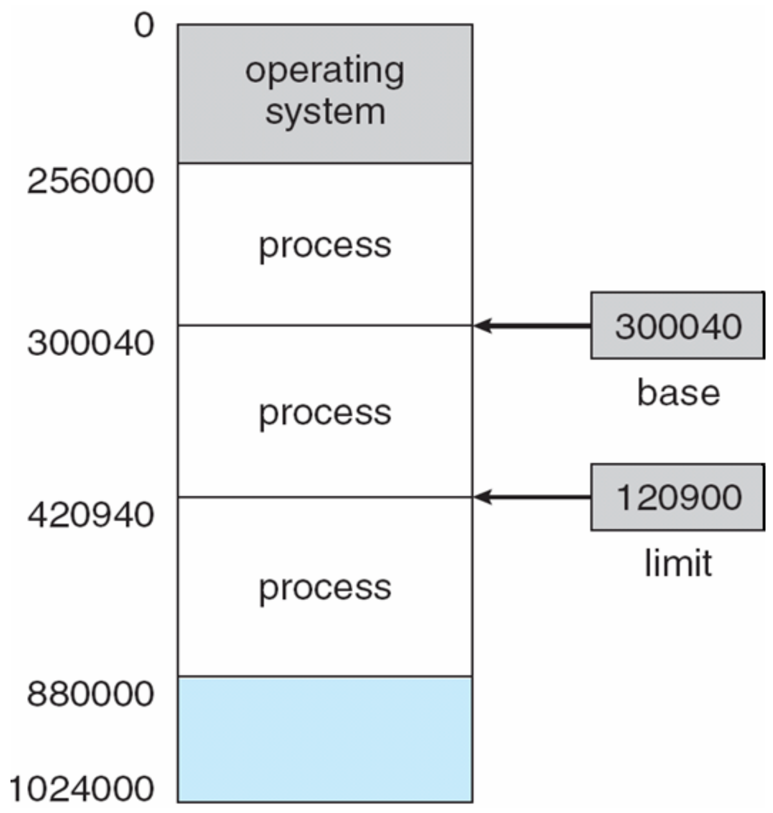
Memory Management

Background

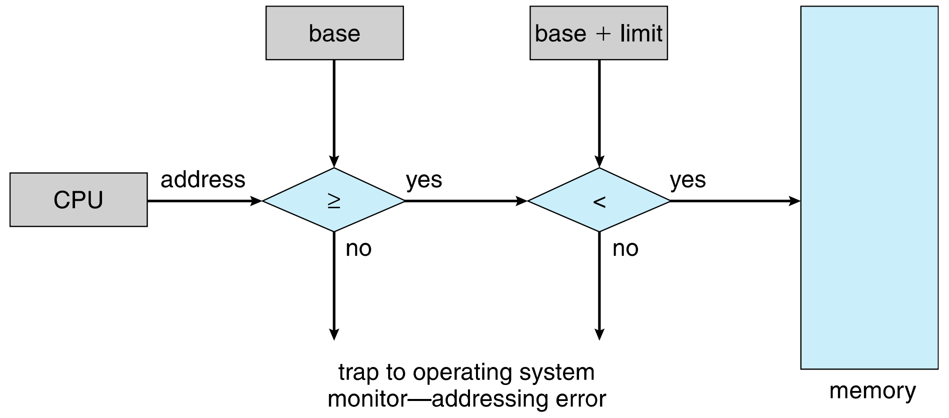
- Memory Management scheme for a specific system depends on many factors, especially on the hardware design of the system.
- Memory is central to the operation of a modern computer system.
- Memory is a large array of words or bytes, each with its own address.
- The CPU fetches instructions from memory according to the value of the program counter. These instructions may cause additional loading from and storing to specific memory addresses.
- 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 in one CPU clock (or less)
- Main memory can take many cycles, causing a **stall**
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

Base and Limit Registers

- A pair of **base** and **limit registers** define the logical address space
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user



Hardware Address Protection



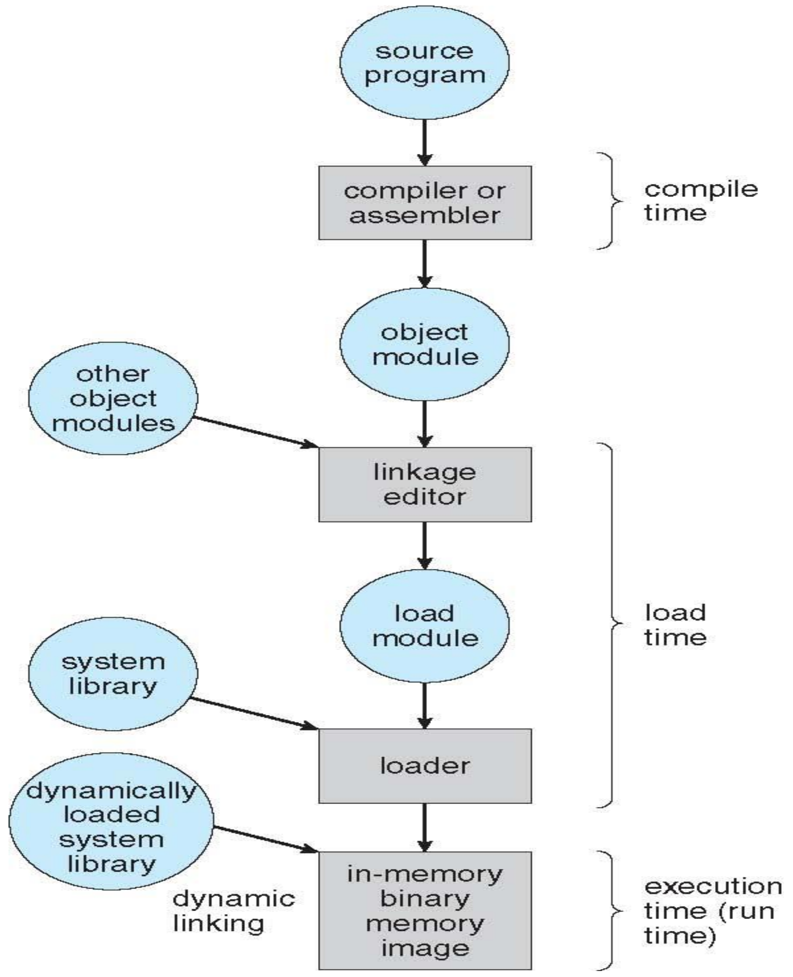
Address Binding

- Usually, a program resides on a disk as a binary executable file.
- The program must be brought into memory and placed within a process for it to be executed.
- Depending on the memory management in use, the process may be moved between disk and memory during its execution.
- The collection of processes on the disk that are waiting to be brought into memory for execution forms the input queue.
- The normal procedure is to select one of the processes in the input queue and to load that process into memory for execution forms the input queue.

Binding of Instructions and Data to Memory

- 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**: If it is not known at compile time where the process will reside in memory, then relocatable code can be generated.
  - Must generate relocatable code if memory location is not known at compile 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

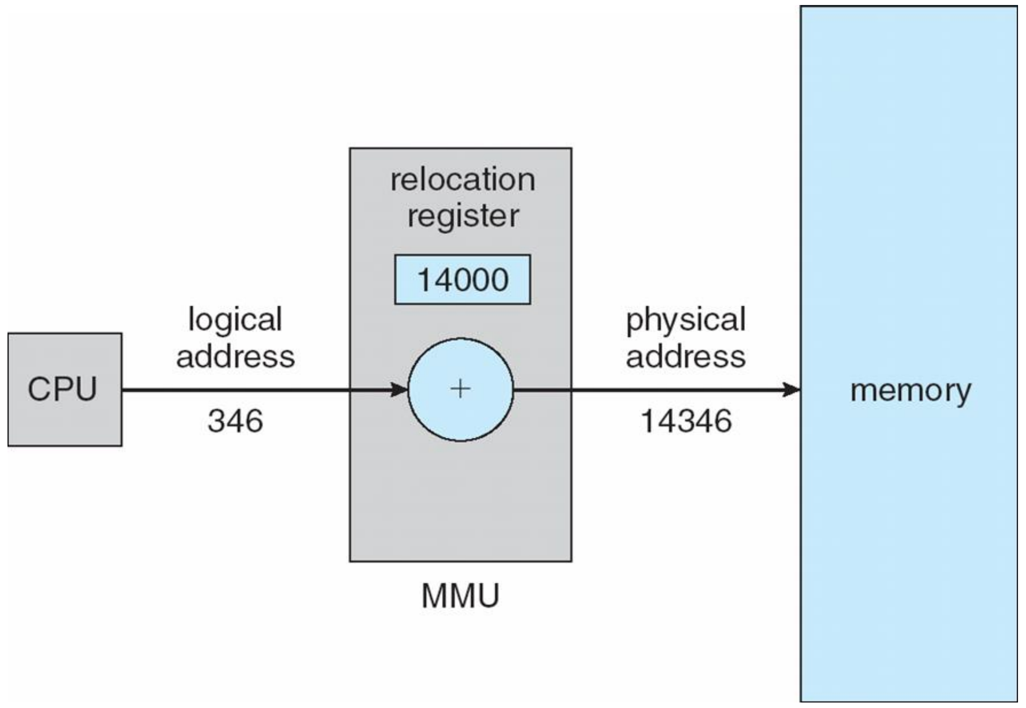
- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management
  - Logical address – generated by the CPU; also referred to as virtual address
  - Physical address – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; 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 to physical address
- Many methods possible, covered in the rest of this chapter
- To start, consider simple scheme where the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
    - Base register now called **relocation register**
    - MS-DOS on Intel 80x86 used 4 relocation registers
- 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

Dynamic relocation using a relocation register

- 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
- No special support from the operating system is required
  - Implemented through program design
  - OS can help by providing libraries to implement dynamic loading



Reference

- Silbersachatz and Galvin, “ Operating System Concepts”, Person

THANKS