جامعة ألأنبار

كلية علوم الحاسوب وتكنولوجيا المعلومات

قسم أنظمة شبكات الحاسوب

المرحلة الرابعه

Operating System

التدريسي: أ.م.د عمر منذر حسين

Introduction

- The Completing a memory access may take many cycles of the CPU clock
- The Separate per-process memory space protects the processes from each other and is fundamental to having multiple processes loaded in memory for concurrent execution.
- To separate memory spaces, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses.
- We can provide this protection by using two registers, usually a **base** and a **limit**, as illustrated in Figure 8.1.
- The **base register** holds the smallest legal physical memory address; the **limit register** specifies the size of the range.
- 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).

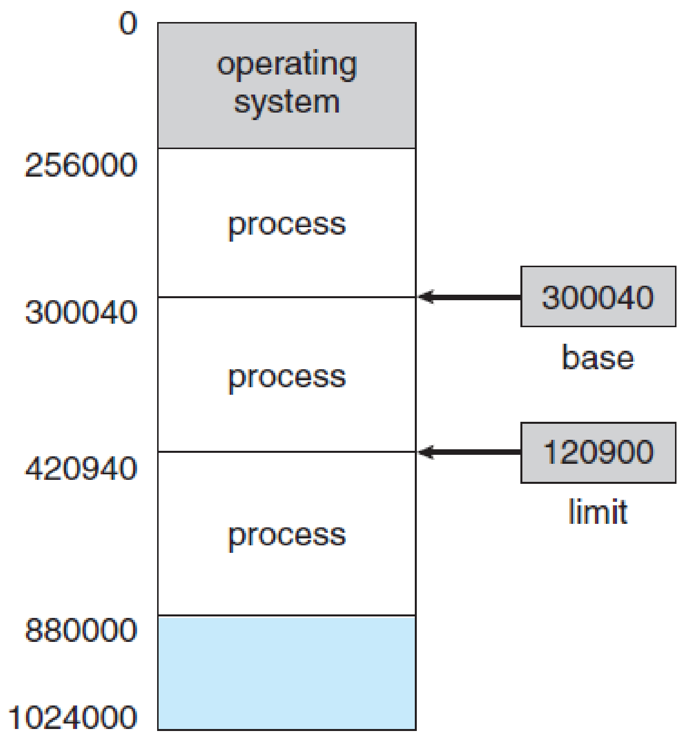

Figure 8.1 A base and a limit register define a logical address space.

- Protection of memory space is accomplished by having the CPU hardware compare every address generated in user mode with the registers.

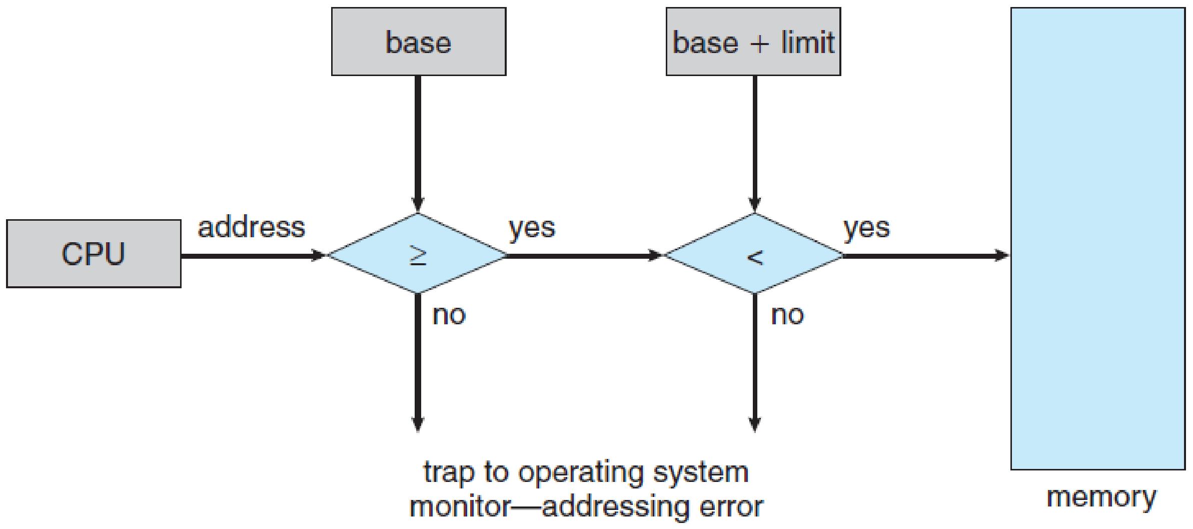

Figure 8.2 Hardware address protection with base and limit registers.

Address Binding

- Most systems allow a user process to reside in any part of the physical memory.
- In most cases, a user program goes through several steps before being executed (Figure 8.3).
- Addresses may be represented in different ways during these steps.
- Addresses in the source program are generally symbolic (such as the variable count).
- A ***compiler*** typically **binds** these symbolic addresses to relocatable addresses.
- The ***linkage editor or loader*** in turn binds the relocatable addresses to absolute addresses (such as 74014).
- Each binding is a mapping from one address space to another.

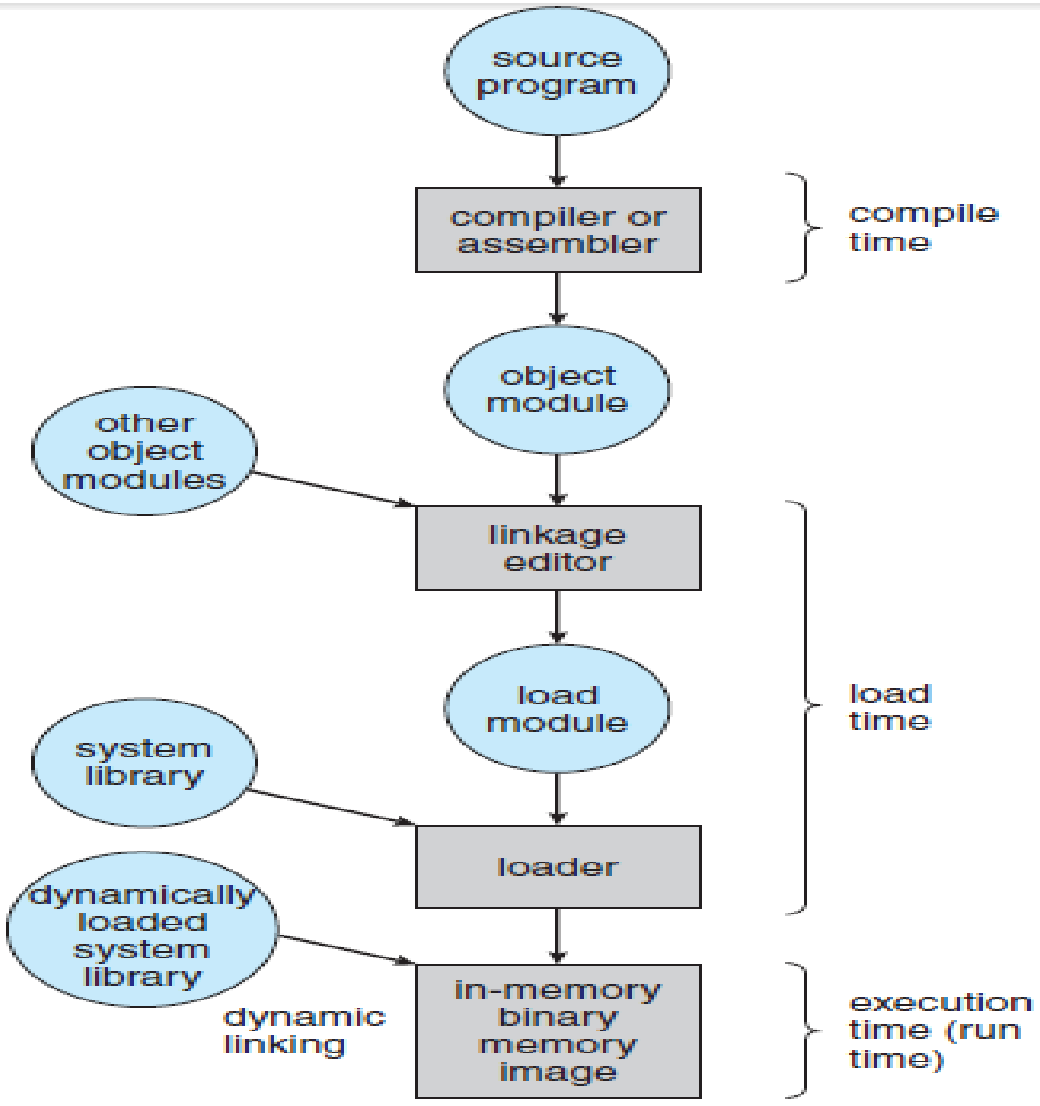

Figure 8.3 Multistep processing of a user program.

Logical Versus Physical Address Space

- An address generated by the CPU is commonly referred to as a **logical address.**
- **A**n address seen by the memory unit—that is, the one loaded into the **memory-address register** of the memory—is commonly referred to as **a physical address.**
- The set of all logical addresses generated by a program is a **logical (virtual) address space**.

- The set of all physical addresses corresponding to these logical addresses is a **physical address space**.
- Thus, in the execution-time address-binding scheme, the logical and physical address spaces differ.
- The run-time mapping from virtual to physical addresses is done by a hardware device called the **memory-management unit (MMU).**
- The base register is called a **relocation register.**
- The value in the relocation register is added to every address generated by a user process at the time the address is sent to memory (see Figure 8.4).
- The user program never sees the real physical addresses.

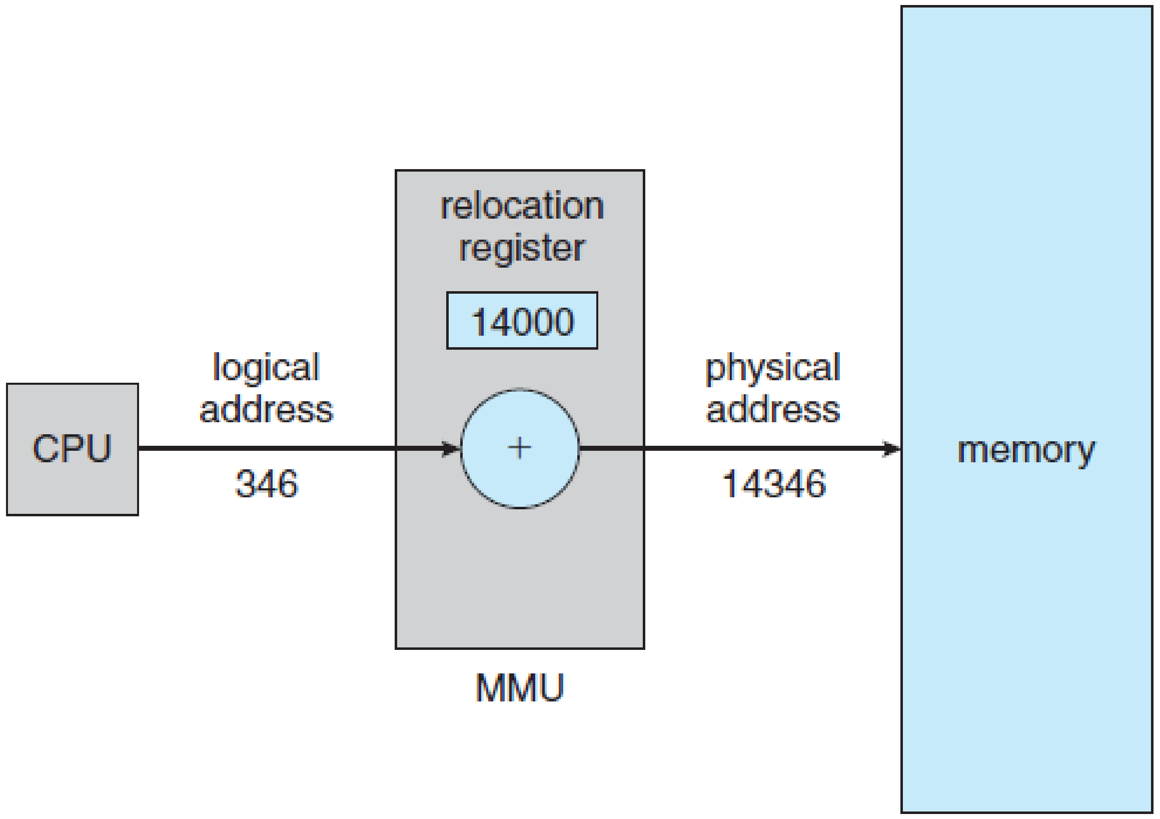

Figure 8.4 Dynamic relocation using a relocation register.

Dynamic Loading

- The size of a process is limited to the size of physical memory.
- To obtain better memory-space utilization, we can use **dynamic loading.**
- With dynamic loading, a routine is not loaded until it is called.
- All routines are kept on disk in a relocatable load format.
- The main program is loaded into memory and is executed.
- When a routine needs to call another routine, the calling routine first checks to see whether the other routine has been loaded.
- If it has not, the relocatable linking loader is called to load the desired routine into memory and to update the program's address tables to reflect this change.

Memory partitioning

- One of the simplest techniques for memory management.
- The user part of the memory must be subdivided to accommodate multiple processes.
- Includes fixed partitioning and dynamic partitioning.
- **Fixed Partitioning:**
- OS occupies some fixed portion of the main memory and the rest of main memory is available for use by multiple processes.
- Partition sizes: Equal-size partitions and unequal size partitions.

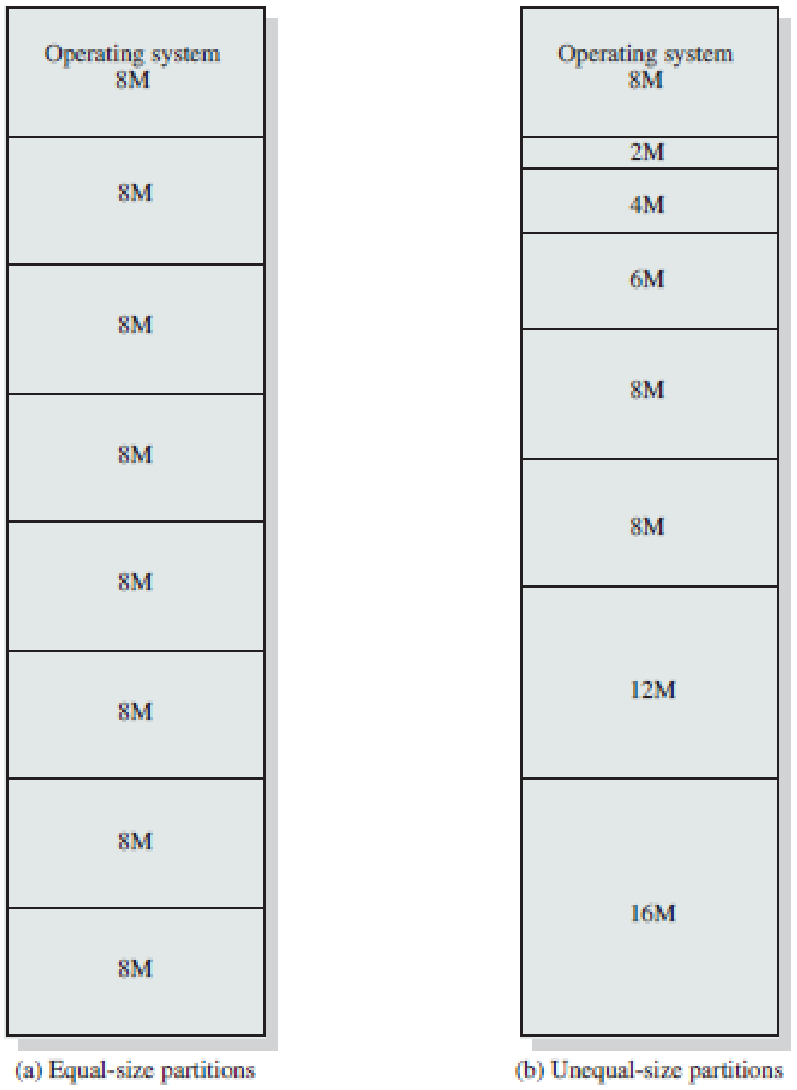

Figure 7.2 **Example of Fixed Partitioning of a 64-Mbyte Memory**

- Any process whose size is less than or equal to partition size can be loaded into any available partition.
- Each partition may contain exactly one process.
- Thus, the degree of multiprogramming is bound by the number of partitions.
- When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.
- Program bigger than partition size!

- Overlay: only a portion of a program need be in a main memory at any one time.
    - **Inefficiency**. Any program no matter how small occupies an entire partition (Internal fragmentation) (wasted space internal to the partition).
    - Unequal-size partition can lessen those problems.

- **Dynamic Partitioning:**
- The partitions are of variable length and numbers.
- The process allocated exactly as much memory as it requires.
- Example of using 64 Mbyte in dynamic partitioning.

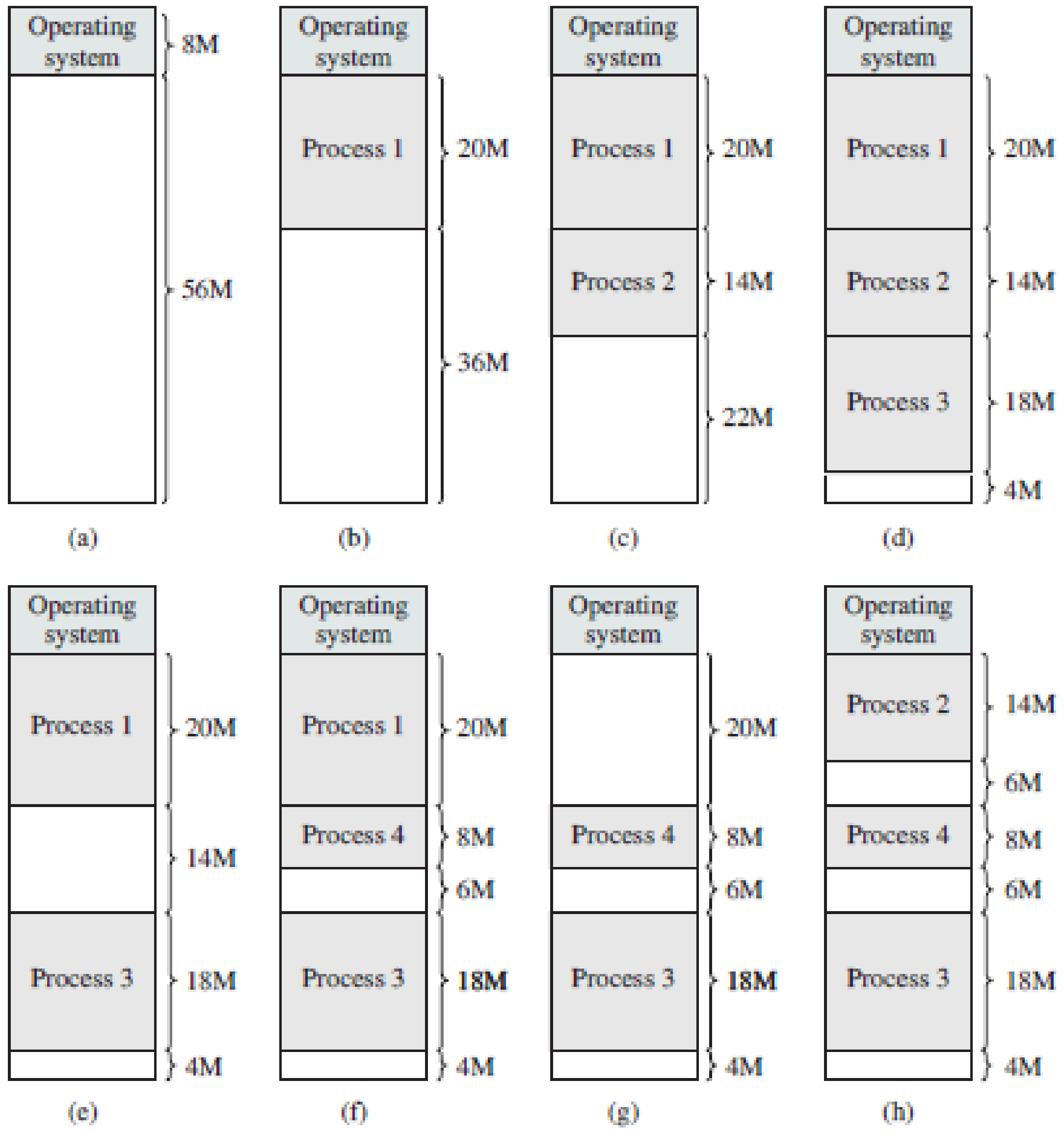

Figure 7.4 The Effect of Dynamic Partitioning

- D: small hole.
- E: swap out process 2.
- F: another small hole.
- G: Swap out process 1.
- End with a lot of small holes in memory.
- **External Fragmentation: memory that is external to all partitions becomes increasingly fragmented.**

- **Compaction is used to overcome external fragmentation**
- OS shift the processes from time to time to be contiguous. (time consuming).
- Compaction needs dynamic relocation capability: move program from one region to another in memory without invalidating the memory referencing in the program.

Memory Allocation

- One of the simplest methods for allocating memory is to divide memory into several fixed-sized **partitions**.
- There are many solutions to select free **holes** from available holes in memory:
- **First-fit**: Allocate the ***first*** hole that is big enough.
  - Can start searching at beginning of set of holes **or** where last first-fit search ended.
  - Stop when sufficient size is encountered.
- **Best-fit**: Allocate the ***smallest*** hole that is 'just' big enough
  - In Best-Fit, we must search entire list, **unless** list is ordered by size.
  - Best-Fit produces the smallest leftover hole.
- **Worst-fit**: Allocate the ***largest*** hole; must also search entire list (unless list is sorted).

Quiz

- The diagram shows an example of memory configuration (dynamic partitioning) after number of placement and swapping out operations have been carried out. **Addresses go from left to right**. Grey areas are processes while the white ones indicate the free memory blocks. The last process placed is (P6).

P1	4 M	P5	1 M	P6	5M	P3	8M	P2	2 M	P4	4M	P7	3M	P8

-

- A new process (P9) (3M Byte) allocation request must be satisfied next. Indicate the intervals of memory where partitions will be created for new process using (First-fit, best-fit, next-fit) algorithms. For each algorithm, draw a horizontal segment under the memory strip and label it clearly. Also, determine the size of generated fragment (If it is generated) after each placement algorithm.