* Chapter 06 : Memory Management *

- Memory management strategies
- Background
- Swapping
- Contiguous memory allocation
- Paging
- Structure of Page Table
- Segmentation

Memory Management Strategies

The memory-management algorithms vary from a primitive bare-machine approach to paging and segmentation strategies. Each approach has its own advantages and disadvantages. Selection of a memory-mgmt method for a specific system depends on many factors, especially on the hardware design on the system. Many algorithms require hardware support, leading many systems to have closely integrated hardware and OS memory management.

Background

In a typical instruction cycle,

- First fetches an instruction from memory
- The instruction is then decoded and may cause operands to be fetched

from memory
- After the instruction has been executed on the operands, results may be stored back in memory.

The memory unit sees only a stream of memory addresses; it does not know how they are generated or what they are for. [Accordingly, we can ignore how a program generates a memory address.]

Registers that are built into the CPU are generally accessible within one cycle of the CPU clock. The same cannot be said of main memory, which is accessed via a transaction on the memory bus. Completing a memory access may take many cycles of the CPU clock. In such cases, the processor normally needs to stall, since it does not have the required data to complete the instruction that it is executing. Cache memory is sure a remedy to add fast memory between CPU and main memory. To manage a cache built into the CPU, the hardware automatically speeds up memory access without any OS control.

Memory Protection: Separate per-process memory space protects the processes from each other and is fundamental to having multiple processes loaded in memory for

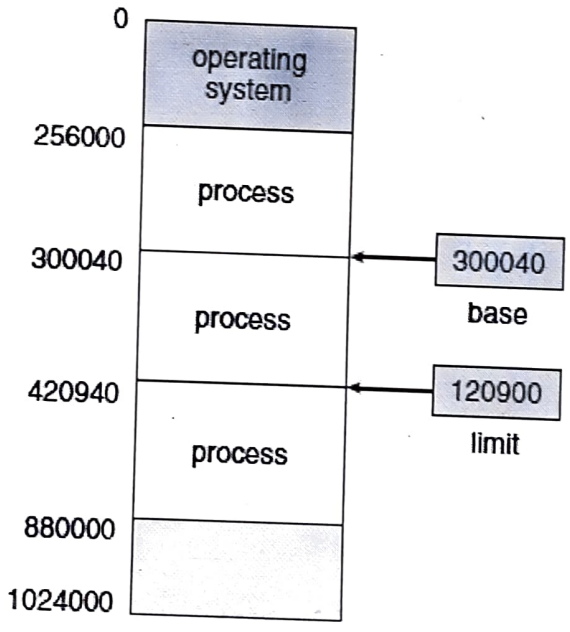
Concurrent execution. Every process must have its own legal addresses that it can access. This is done through base and limit register.

Base register - holds the smallest legal physical address

Limit register - specifies the range

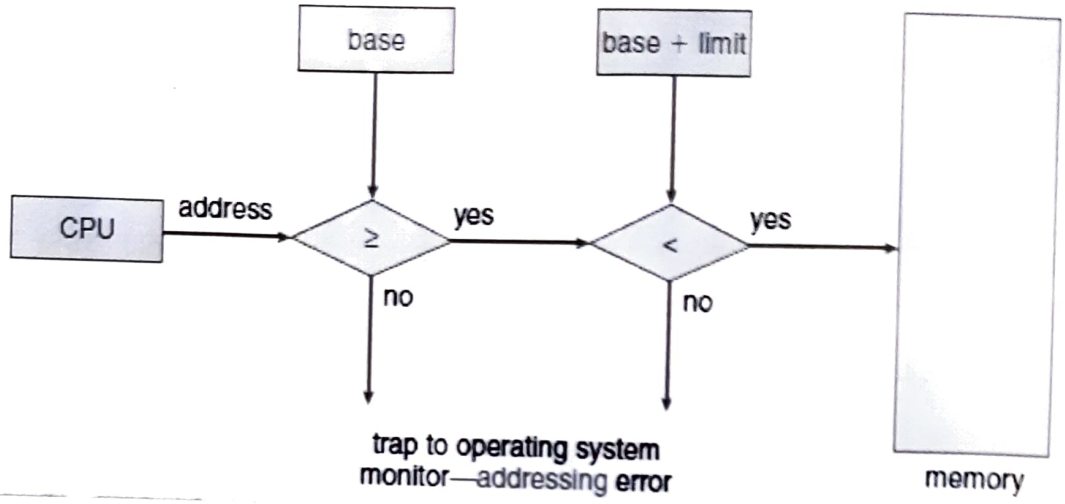
Example: If base = 300040
limit = 120900

Then program can legally access all addresses from 300040 through 420939 (inclusive).



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

Hardware Protection



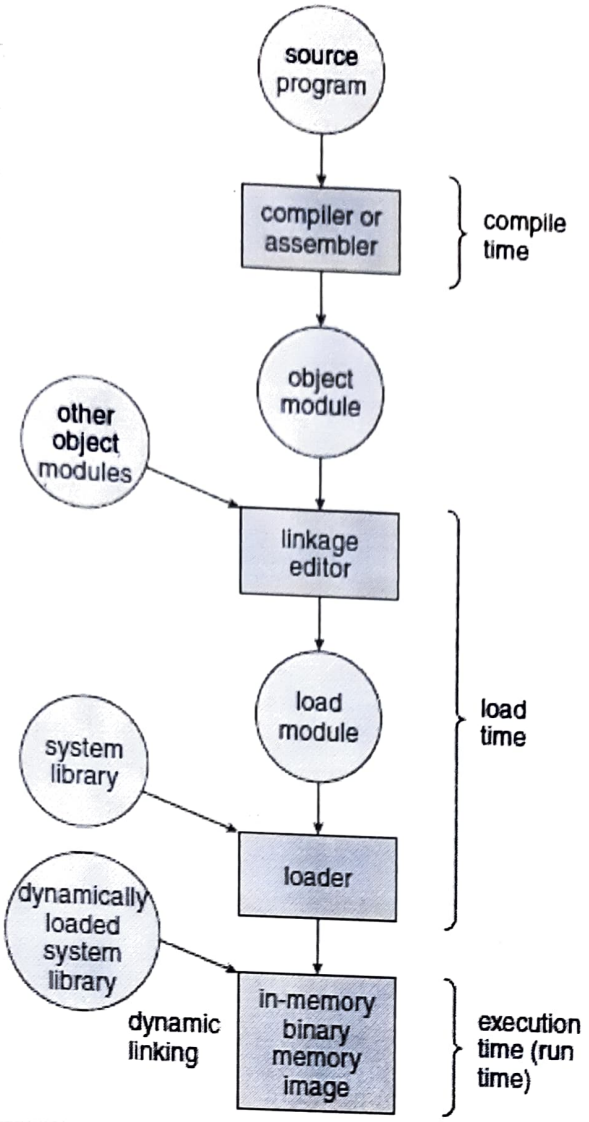
As seen above, any illegal memory access leads to a trap to the OS.

The base and limit registers can be loaded only by the OS, which uses a special privileged instruction. As privileged instructions can be executed only in kernel mode, and since only the OS executes in kernel mode, only the OS can load the base and limit registers.

Address Binding:

The processes on the disk that are waiting to be brought into memory for execution form the input queue.

Most systems allow a user process to reside in any part of the physical memory, and the first address of the user process need not be 0000.



Multistep Processing of a User Program

In most cases, a user program goes through several steps before being executed. Addresses get represented in different ways during these steps.

- Addresses in source program are generally symbolic (like variable-count)
- A compiler typically binds these symbolic addresses to relocatable addresses (like 20 bytes from beginning of this module)
- The linkage editor or loader binds relocatable to absolute addresses (like 74014)

The binding of instructions and data to memory addresses can be done at any step along the way:

Compile time - If you know at compile time where the process will reside in memory, then absolute code can be generated. If starting location changes, the code needs to be re-compiled.

Load time - If not known at compile time, compiler generates relocatable code. In this case the final binding is delayed until load time. If starting address changes, we only need to reload the user code to incorporate the changed value.

Execution time - If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time. The address maps are supported by the hardware.

Logical v/s Physical Address Space

An address generated by CPU is commonly referred to as a Logical address and address seen by the memory unit, that is, the one loaded into the memory-address register of the memory is

referred as a physical address

The compile-time and load-time address-binding methods generate identical logical and physical addresses. However, the execution-time address-binding scheme results in different logical & physical addresses.

Address-binding is the process of mapping from one address space to another address space.

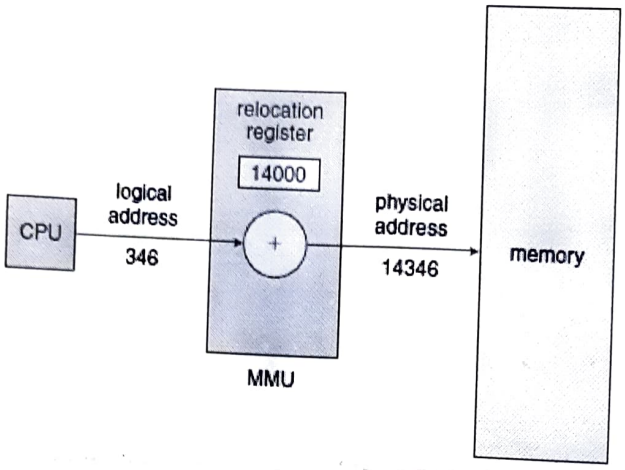
When logical & physical addresses differ, logical address is also referred as virtual address.

The set of all phy logical addresses generated by a program is called as a logical address space.
The set of all physical addresses corresponding to these logical addresses is a physical address space.

The runtime mapping from virtual to physical addresses is done by a hardware device called the

memory-management unit.

which happens through a relocation register.



Dynamic relocation using a relocation register

The value of relocation register is added to every address generated by a user process at the time the address is sent to memory.

Example: If the relocation register is 14000, then access to 346 is mapped to location 14346. The user program never sees the real physical address.

The user program generates only logical addresses and thinks that the process runs in location 0 to max. However these logical addresses must be mapped to physical addresses before they are used. The Concept of a logical address space that is bound to a separate physical address space is central to proper memory management.

Parameter	LOGICAL ADDRESS	PHYSICAL ADDRESS
Basic	generated by CPU	location in a memory unit
Address Space	Logical Address Space is set of all logical addresses generated by CPU in reference to a program.	Physical Address is set of all physical addresses mapped to the corresponding logical addresses.
Visibility	User can view the logical address of a program.	User can never view physical address of program.
Generation	generated by the CPU	Computed by MMU
Access	The user can use the logical address to access the physical address.	The user can indirectly access physical address but not directly.

Dynamic Loading

- With dynamic loading, a routine is loaded only when it is needed

- All routines are kept on disk in a relocatable load format. The main program is loaded into memory and 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 table to reflect this change.
Then the control is passed to the newly loaded routine.

- dynamic loading leads better space utilization

- dynamic loading does not require special support from the OS.

Dynamic Linking & Shared Libraries

- A static linking process combines System libraries along with all other object modules while generating program binary image. In dynamic linking it is postponed until execution time.

- Small piece of code, stub, is used to locate the appropriate memory-resident library routine

- Stub replaces itself with the address of the routine & executes the routine

- OS will take the responsibility to check if the routine is in processes memory address

- Dynamic linking is particularly useful for libraries.

- This system is also called as Shared libraries.