Memory Management

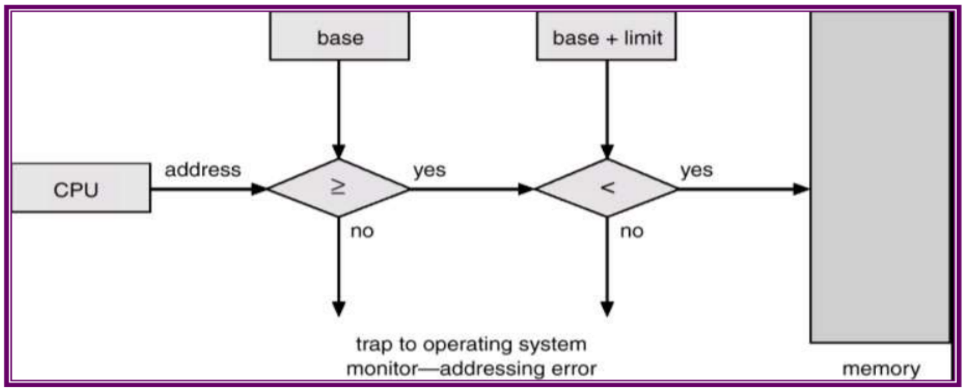
Need?

We need a memory management scheme to manage several processes in the main memory
And Main memory is very small can not reside everything at once

Basic Hardware requirement:

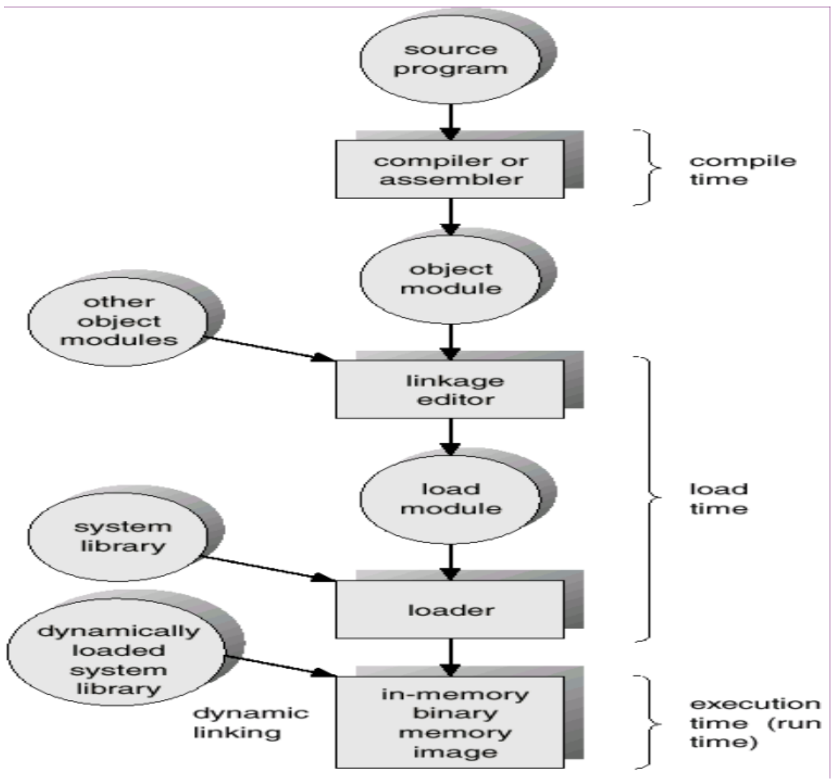
- Ensuring correct operation Each process has a *separate memory space*
- *Base and limit registers* Every address should be compared
    - Base register – holds the smallest legal physical memory address.
    - Limit register – contains the size of the range

Hardware Address Protection



Address Binding:

- A binding is a mapping from one address space to another
- How?
    - Addresses in the source program are symbolic
    - A compiler will bind these addresses to re-locatable addresses.
    - a loader will bind these re-locatable addresses to absolute addresses.
- Types?
    - Compile time:
        - If at compile time memory location is known.
        - absolute code can be generated; must recompile code if starting location changes.
    - Load time:
        - The compiler must generate relocatable code if the memory location is not known at compile time.
        - Final binding is delayed until load time.
        - If the starting address changes, we need only to reload.
    - 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 (base and limit reg)



Dynamic loading:

- Used to obtain better **memory-space** utilization
- unused routine is never loaded
- **The routine** is not loaded until it is called
- Useful when large amounts of code are needed to handle infrequently occurring cases. (will not occupy or block space)
- No special support from the operating system is required. Implemented through program design
- Dynamic loader is very much flexible because it can adapt to the changes in the program's requirements at dynamic environment (change in version of printf).
- reduce the executable file size by excluding the libraries
- But it is complex to implement

Dynamic Linking:

- linking is postponed until the execution time
- Smaller file size, as libraries are linked dynamically at runtime.
- More flexible, as libraries can be updated or replaced without recompiling the program.
- A 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
- Dynamic linking is particularly useful for libraries.

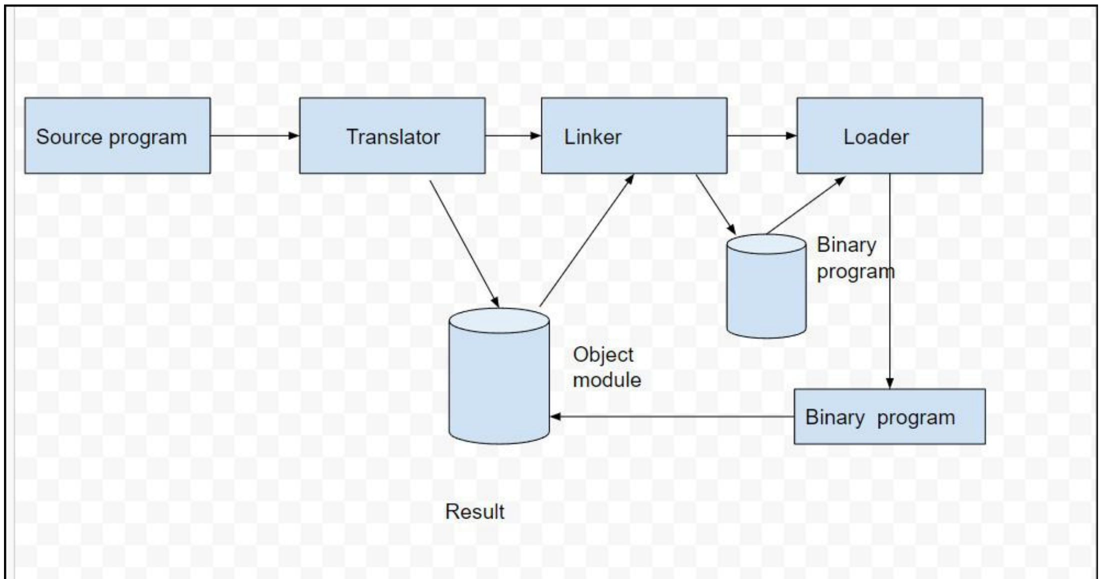
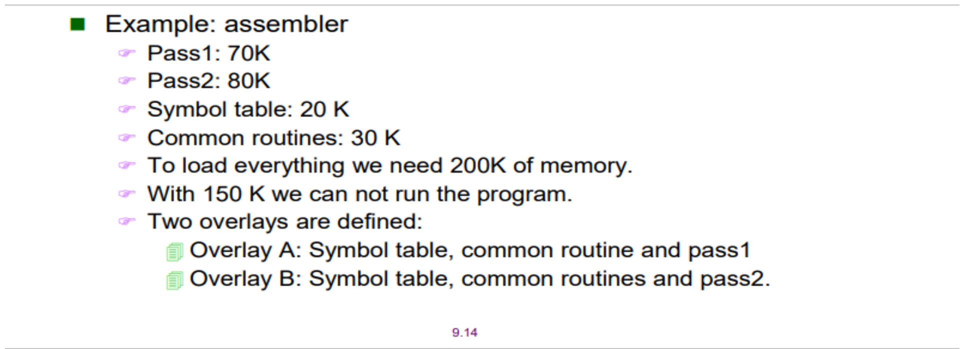


Fig : Dynamic Linking

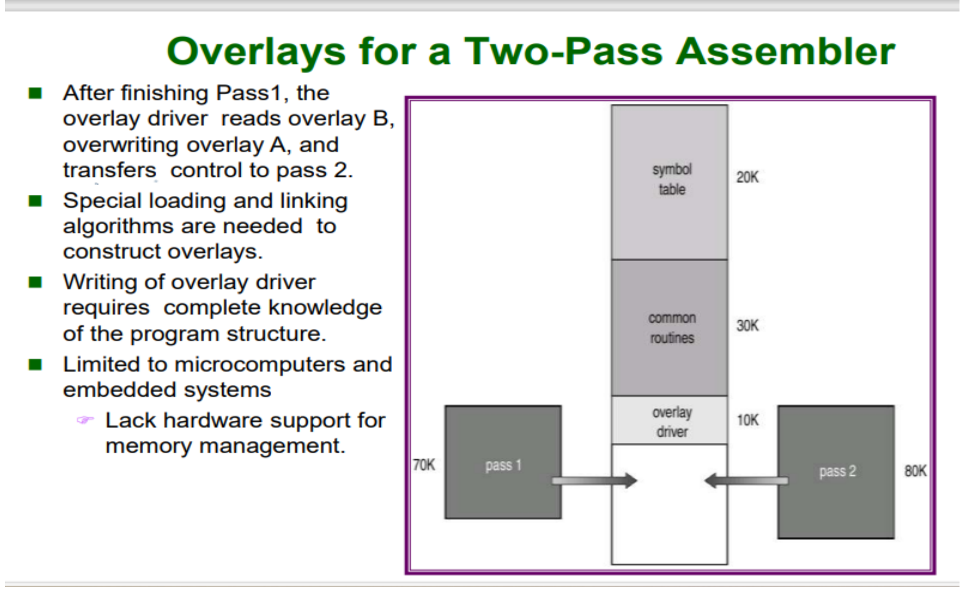
Overlays:

- Used when a process to be larger than the main memory.
- Keep in memory only those instructions and data that are needed.
- Needed when process is larger than amount of memory allocated
- Write below example



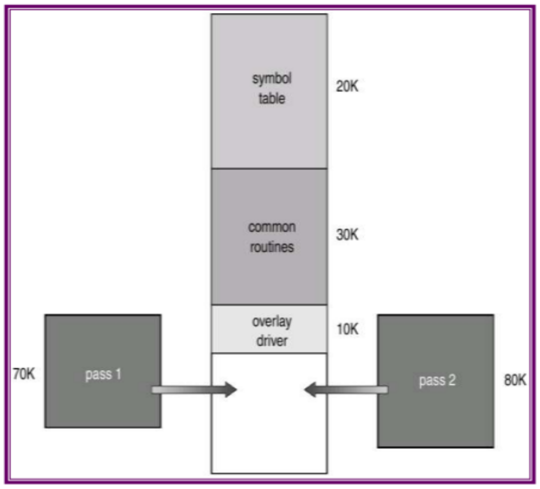
- Example: assembler
  - Pass1: 70K
  - Pass2: 80K
  - Symbol table: 20 K
  - Common routines: 30 K
  - To load everything we need 200K of memory.
  - With 150 K we can not run the program.
  - Two overlays are defined:
    - Overlay A: Symbol table, common routine and pass1
    - Overlay B: Symbol table, common routines and pass2.

9.14



Overlays for a Two-Pass Assembler

- After finishing Pass1, the overlay driver reads overlay B, overwriting overlay A, and transfers control to pass 2.
- Special loading and linking algorithms are needed to construct overlays.
- Writing of overlay driver requires complete knowledge of the program structure.
- Limited to microcomputers and embedded systems
  - Lack hardware support for memory management.

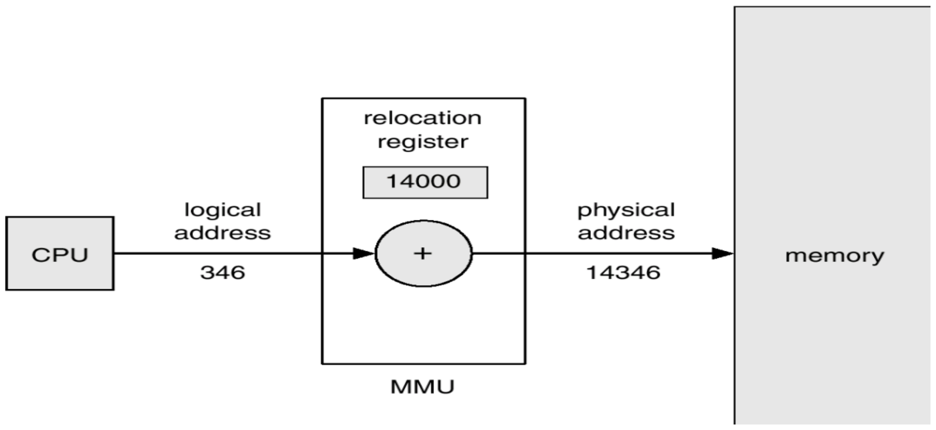


Address spaces

- Logical address – generated by the CPU; also referred to as virtual address.
    - Set of logical addresses generated by a program is called **logical address space.**
- Physical address – address seen by the memory unit.
    - Set of physical addresses corresponds to logical space is called **physical address space**

MMU:

- Hardware device that maps virtual to physical address
- Relocation Register is added to every address generated by a user process at the time it is sent to memory



SWAPPING:

- Whenever the CPU scheduler decides to execute a process, it calls the dispatcher.
- The dispatcher checks whether the process is in main memory.
- If the process is not, and there is no free memory, the dispatcher swaps out a process currently in main memory and swaps in the desired process.
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped.
- Problems:
  - A process should be idle to swap. It should not have any pending I/Os.
    - Solutions Never swap a process with pending I/Os
    - Execute I/O operations into OS buffers.

w