OPERATING SYSTEMS

Memory Management

Chandravva Hebbi

Department of Computer Science

OPERATING SYSTEMS

Main Memory: Hardware and control structures, OS support, Address translation

Chandravva Hebbi

Department of Computer Science

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Background

- What is a memory?
- Memory consists of a large array of bytes, each with its own address.
- Execution of an instruction.
  - Fetch an Instruction from memory, Decode the instruction, operands are fetched(from memory or registers)
  - After the instruction is executed, results are stored back.
- The memory unit(MU) sees the stream of addresses.
- Memory unit does not know how these addresses are generated.
- We will learn how the addresses are generated by the running program.

Basic Hardware

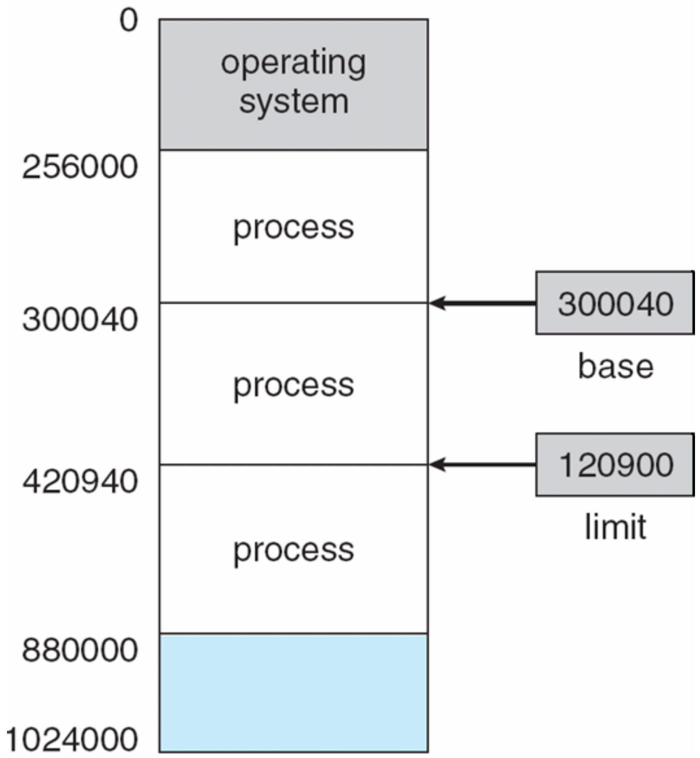
- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- The data required for CPU must be made available in the registers.
- Register access in one CPU clock (or less)
- Main memory can take many cycles, causing a **stall**
- **Cache** sits between main memory and CPU registers
  - Speeds up memory access without any OS control
- Protection of memory required to ensure correct operation

Basic Hardware

- Protection of OS from its access by user processes
- On multiuser systems user processes must be protected from each other.
- This protection must be provided by the hardware because the operating system doesn't usually intervene between the CPU and its memory accesses.
- Several hardware protection methods will be discussed.
- Protection by using two registers, usually a base and a limit.

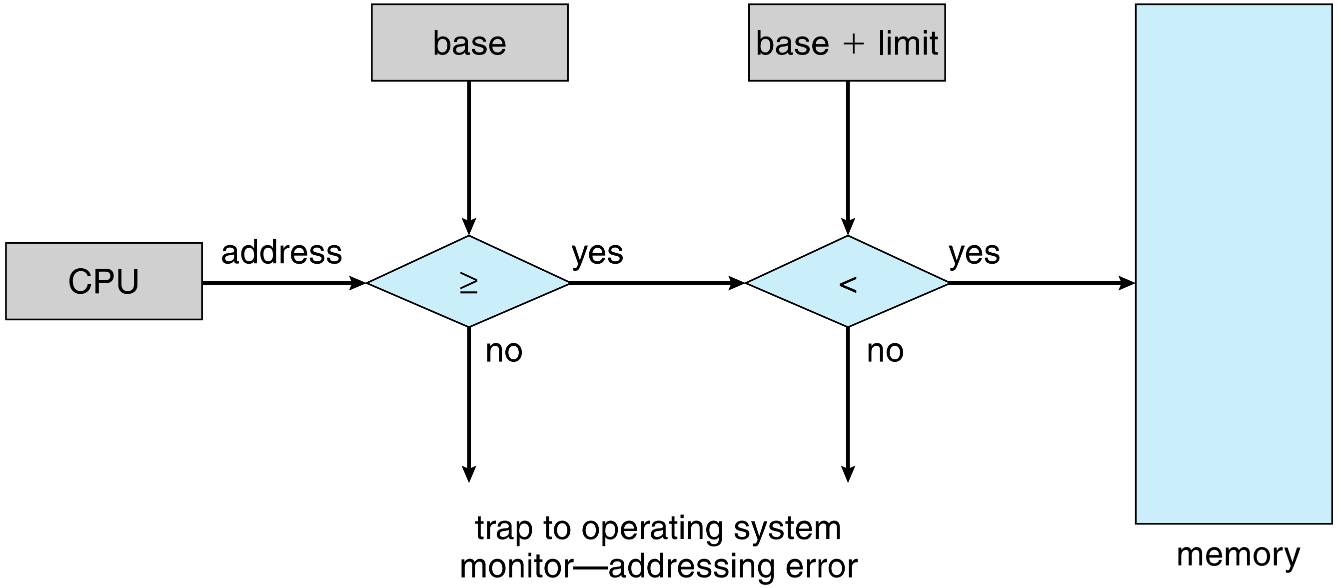
Basic Hardware

- 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

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



- prevents a user program from modifying the code or data structures of either the operating system or other users.

- The base and limit registers can be loaded only by the operating system.
  - Using special privileged instruction.
  - privileged instructions can be executed only in kernel mode.
  - OS runs in the kernel mode. Only OS can change the register values.
  - This prevents other user programs to modify the register values.

Address Binding

- Programs on disk as binary executable and are brought to main memory for execution
- Processes may be moved between the disk and memory during execution.
- What are the steps involved in the program execution?
- 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 but the first address of the user process need not be 00000
- Addresses in the source program are generally symbolic (Ex. variable count).

Address Binding

- A compiler typically binds these symbolic addresses to relocatable addresses
  - “14 bytes from the beginning of this module”
- The linkage editor or loader in turn binds the relocatable addresses to absolute addresses
  - Ex. 74014
- Each binding is a mapping from one address space to another.

Memory-Management Unit (Cont.)

Base-register scheme for address mapping

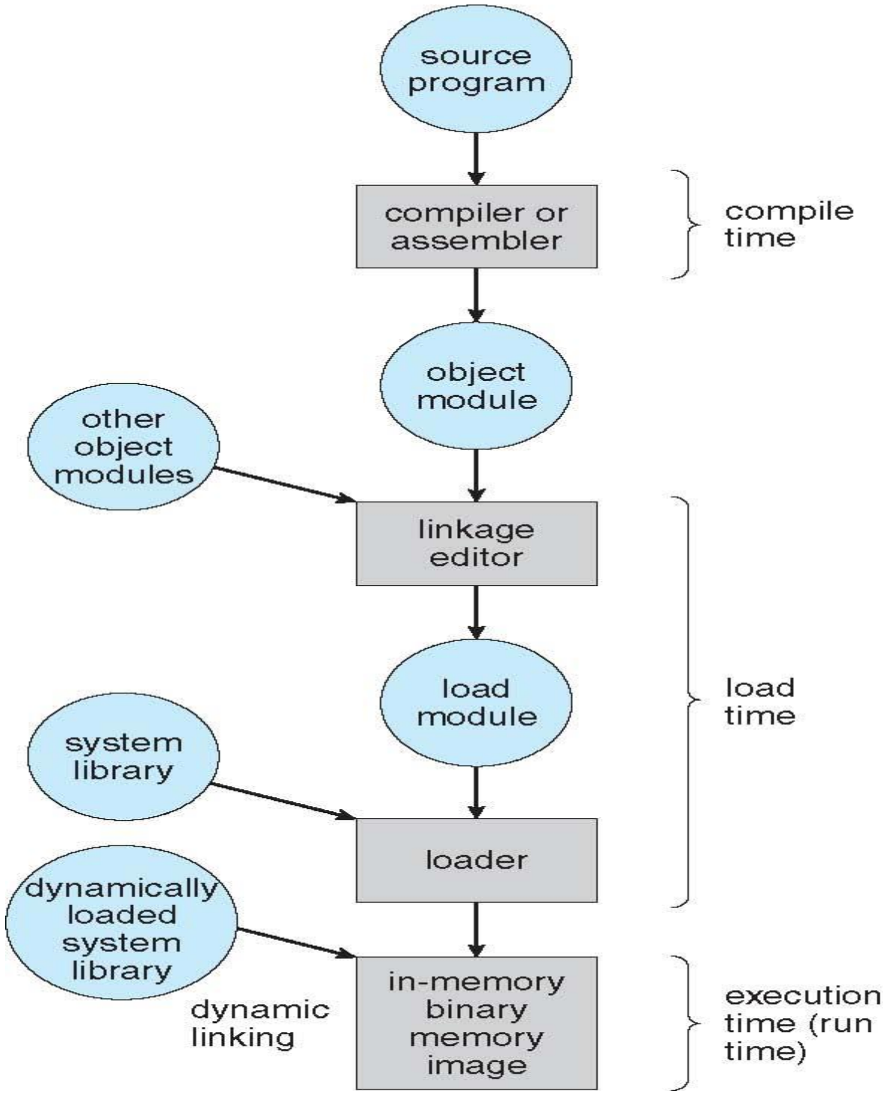
- The base register also referred to as called a relocation register.
- The value in the relocation register is added to every address generated by a user process at the time the address is sent to memory .
- Example, if the base is at 14000, then an attempt by the user to address location 0 is dynamically relocated to location 14000;
- An access to location 346 is mapped to location 14346.
- 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.)

The binding of instructions and data to memory addresses

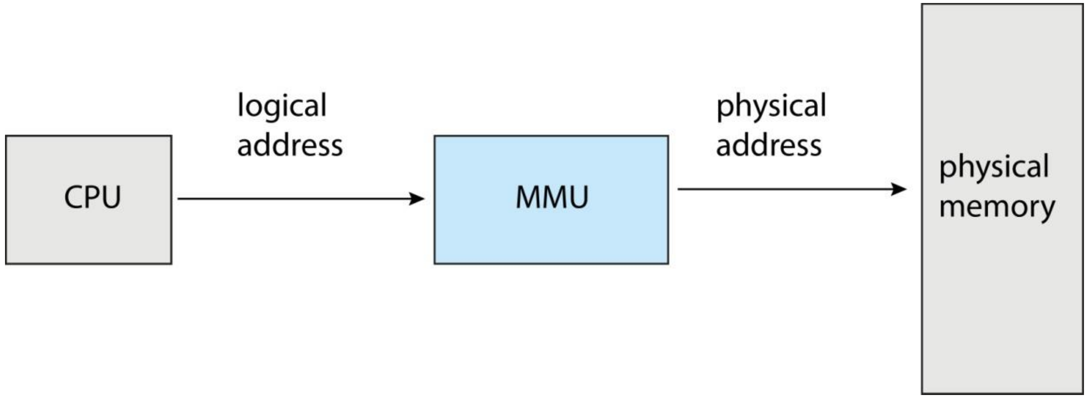
- **Compile time**. If you know at compile time where the process will reside in memory, then **absolute code** can be generated.
  - If the start address changes, it is necessary to recompile once again
  - The MS-DOS .COM-format programs are bound at compile time.
- **Load time**. If it is not known at compile time where the process will reside in memory, then the compiler must generate **relocatable code**.
  - final binding is delayed until load time.
  - If the starting address changes, we need only reload the user code to incorporate this 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.
  - Special hardware must be available for this scheme to work

Multistep Processing of a User Program



Memory-Management Unit (MMU)

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



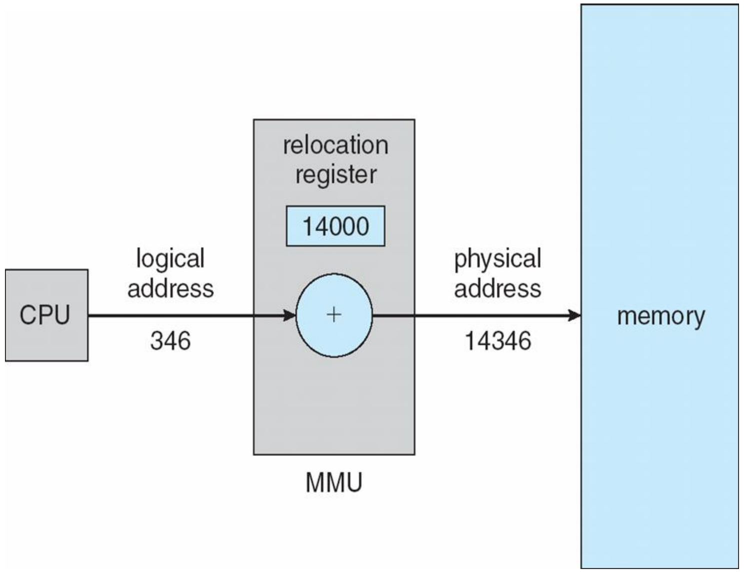
- Many methods possible to accomplish this mapping, will be discussed in the next few lectures.

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

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



Dynamic Linking

- **Static linking** – system libraries and program code combined by the loader into the binary program image
- Dynamic linking –linking postponed until execution time
- 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
- Operating system checks if routine is in processes' memory address
  - If not in address space, add to address space
- Dynamic linking is particularly useful for libraries
- System also known as **shared libraries**
- Consider applicability to patching system libraries
  - Versioning may be needed

- A program whose necessary library functions are embedded directly in the program's executable binary file is ***statically*** linked to its libraries
- The main disadvantage of static linkage is that every program generated must contain copies of exactly the same common system library functions
- *Dynamic* linking is more efficient in terms of both physical memory and disk-space usage because it loads the system libraries into memory only once

Demo

- $cc fork.c
- size a.out
- $cc -static fork.c
- size a.out

THANK YOU

Chandravva Hebbi
Department of Computer Science Engineering
chandravvahebbi@pes.edu