CSC 4103 - Operating Systems
Spring 2007

LECTURE - X

MAIN MEMORY - I

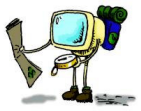
Tevfik Koşar

Louisiana State University

February 27th, 2007

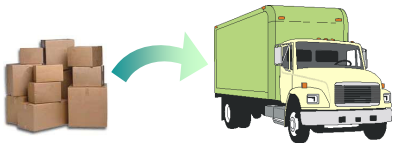
Roadmap

- Memory Management
- Main Memory
- Logical vs Physical Address Space
- Address Protection
- Address Binding
- Dynamic Loading



Memory Management Requirements

- **The O/S must fit multiple processes in memory**
  - ✓ memory needs to be subdivided to accommodate multiple processes
  - ✓ memory needs to be allocated to ensure a reasonable supply of ready processes so that the CPU is never idle
  - ✓ memory management is an **optimization** task under **constraints**



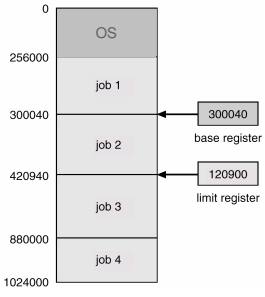
Fitting processes into memory is like fitting boxes into a fixed amount of space

Memory Management Requirements

- **Memory management must satisfy various requirements**
  - ✓ **relocation** of address references
    - must translate memory references to physical addresses
  - ✓ **protection** of memory spaces
    - forbid cross-process references
  - ✓ **sharing** of memory spaces
    - allow several processes to access a common memory area
  - ✓ **logical organization** (of programs)
    - programs are broken up into independent modules
  - ✓ **physical organization** (of memory)
    - fit multiple programs and modules in physical memory

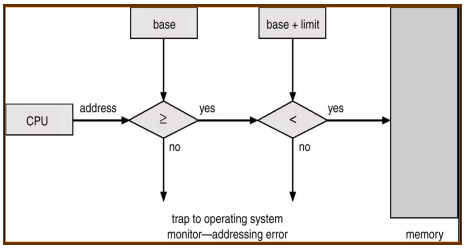
Logical Address Space

- Each process has a separate memory space
- Two registers provide address protection between processes:
- Base register: smallest legal address space
- Limit register: size of the legal range



HW Address Protection

- CPU hardware compares every address generated in user mode with the registers
- Any attempt to access other processes' memory will be trapped and cause a fatal error



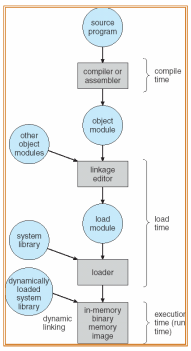
Address Binding

- Addresses in a source program are generally symbolic
  - eg. int count;
- A compiler binds these symbolic addresses to relocatable addresses
  - eg. 100 bytes from the beginning of this module
- The linkage editor or loader will in turn bind the relocatable addresses to absolute addresses
  - eg. 74014
- Each binding is mapping from one address space to another

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**: Must generate *relocatable code* if memory location is not known at compile time; need only reload if starting address changes
- **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.



Logical vs Physical Address Space

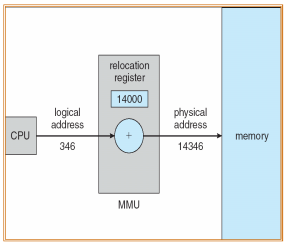
- Address generated by CPU is referred as logical address
- Address seen by the memory is seen as physical address
- Compile time and load time methods generate identical logical and physical addresses
- Execution-time method generates different logical and physical addresses
  - Logical address referred as virtual address

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

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address
- In MMU scheme, the value in the relocation register (base register) is added to every address generated by a user process at the time it is sent to memory
- The user program deals with *logical* addresses; it never sees the *real* physical addresses



Dynamic Loading

- Used to increase memory space utilization
- A routine is not loaded until it is called
  - All routines do not need to be in memory all time
  - Unused routines never loaded
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required to implement
- When a routine needs to call another routine:
  - Caller first checks if that routine is already in memory
  - If not, loader is called
  - New routine is loaded, and program's address tables updated

Any Questions?

13

Reading Assignment

- Read chapter 8 from Silberschatz.

14

Acknowledgements

- "Operating Systems Concepts" book and supplementary material by Silberschatz, Galvin and Gagne.
- Also some slides are borrowed from René Doursat at UNR.

15