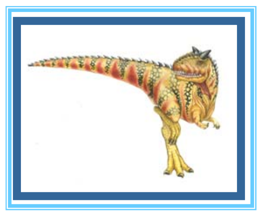
Chapter 8: Main Memory



(slides improved/selected by R. Doemer, 01/14/11)

Chapter 8: Memory Management

- Background
- Swapping
- Contiguous Memory Allocation
- Paging
- Structure of the Page Table
- Segmentation
- Example: The Intel Pentium

(slide modified by R. Doemer, 05/17/10)

Objectives

- To provide a detailed description of various ways of organizing memory hardware
- To discuss various memory-management techniques, including paging and segmentation
- To provide a detailed description of the Intel Pentium, which supports both pure segmentation and segmentation with paging

(slide modified by R. Doemer, 05/17/10)

Background

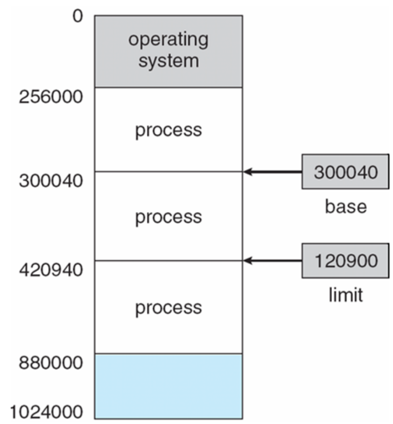
- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are the only storage the CPU can access directly
- Register access in one CPU clock cycle (or less)
- Main memory access can take many cycles
- **Cache** sits between main memory and CPU registers

- Protection of memory is required to ensure safe cooperation of processes

(slide modified by R. Doemer, 05/17/10)

Base and Limit Registers

- A pair of **base** and **limit** registers define the logical address space

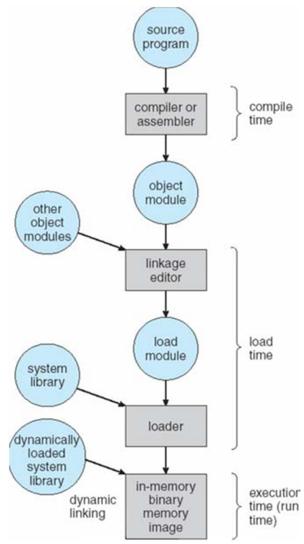


Binding of Instructions and Data to Memory

- **Address binding** of instructions and data to memory addresses can happen at three different stages
  - **Compile time**: If memory location is known a priori, *compiler* can generate **absolute code**; must recompile code if starting location changes
  - **Load time**: Compiler must generate **relocatable code** if memory location is not known at compile time; *Loader* completes address binding
  - **Execution time**: Address binding can be delayed until run time if the process can be moved during its execution from one memory segment to another; need hardware support in CPU for address mapping (e.g., base and limit registers); *Memory Management Unit* in CPU determines address binding

(slide modified by R. Doemer, 05/17/10)

Multi-Step Processing of a User Program



Compile time

Link time

Load time

Run time

(slide modified by R. Doemer, 05/17/10)

Dynamic Linking

- **Dynamic Linking**: Linking is postponed until execution time
- Dynamic linking is also known as **shared libraries**

- A small piece of code, a *stub routine*, is used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and then executes the routine
- Operating system needed to check if routine is in the processes' memory address
- Dynamic linking is particularly useful for libraries (which then can be shared by multiple processes)

(slide modified by R. Doemer, 05/17/10)

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

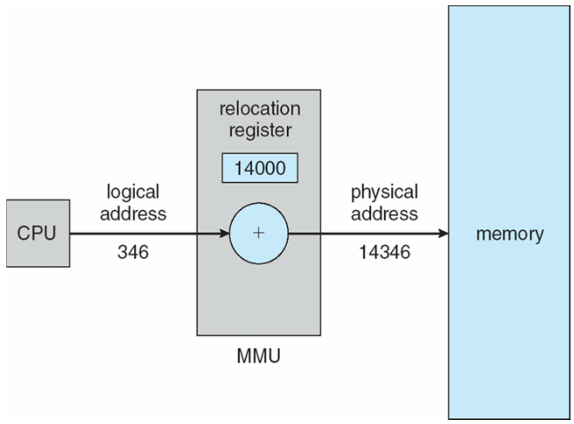
(slide modified by R. Doemer, 05/17/10)

Memory-Management Unit (MMU)

- Memory-Management Unit (MMU): Hardware device that maps virtual to physical address
- In MMU scheme, the value in a **relocation 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

(slide modified by R. Doemer, 05/17/10)

Conceptual MMU with a Relocation Register



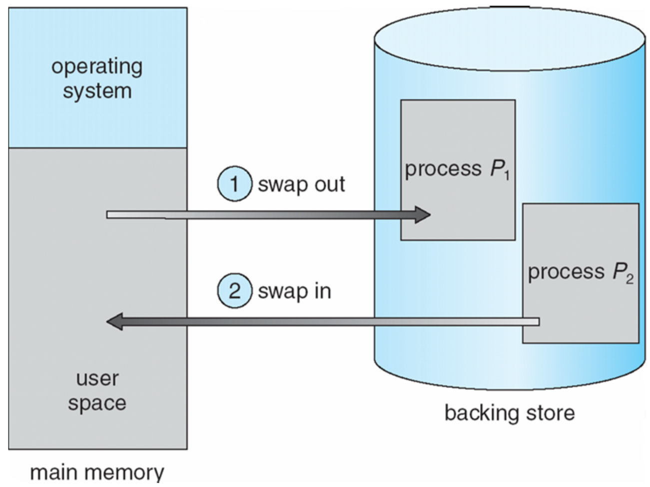
(slide modified by R. Doemer, 05/17/10)

Swapping

- **Swapping**:
  A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
- **Backing store** – fast disk, large enough to accommodate copies of all memory images for all processes;
  must provide direct access to these memory images
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk
- Major part of **swap time** is transfer time;
  transfer time is directly proportional to the amount of memory swapped
- **Roll out, roll in** – swapping variant used for priority-based scheduling;
  lower-priority process is swapped out so that a higher-priority process can be loaded and executed
- Modified versions of swapping are found on many systems
  (i.e., UNIX, Linux, and Windows)

(slide modified by R. Doemer, 05/17/10)

Schematic View of Swapping



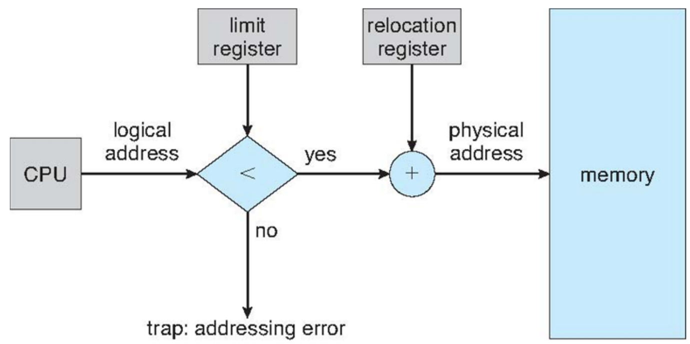
Contiguous Allocation

- Main memory is usually divided into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
- **Relocation registers** are used to protect user processes from each other, and from changing operating-system code and data
  - **Base register** contains value of smallest physical address
  - **Limit register** contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address to physical address *dynamically (at run time)*

(slide modified by R. Doemer, 05/17/10)

Contiguous Allocation

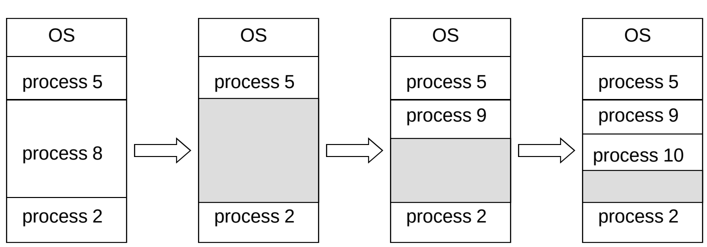
- Hardware Support for Relocation and Limit Registers



(slide modified by R. Doemer, 05/17/10)

Contiguous Allocation

- Multiple-partition allocation
  - **Hole** – block of available memory;
    holes of various sizes are scattered throughout memory
  - When a process arrives, it is allocated memory
    from a hole large enough to accommodate it
  - Operating system maintains information about:
    a) allocated partitions b) free partitions (hole)



(slide modified by R. Doemer, 05/17/10)

Contiguous Allocation

Dynamic Storage-Allocation Problem:
How to satisfy a request of size *n* from a list of free holes

- **First-fit**: Allocate the *first* hole that is big enough
- **Best-fit**: Allocate the *smallest* hole that is big enough
  - Produces the smallest leftover hole
  - Must search entire list, unless ordered by size
- **Worst-fit**: Allocate the *largest* hole
  - Produces the largest leftover hole
  - Must also search entire list, unless ordered by size

First-fit and best-fit are usually better than worst-fit
in terms of speed and storage utilization.

(slide modified by R. Doemer, 05/17/10)

Memory Fragmentation

- **Internal Fragmentation** –
  allocated memory is often slightly larger than requested memory (e.g., 64 bytes allocated for a request of 55 bytes); this size difference is *internal* to a memory partition, but is not being used
- **External Fragmentation** –
  many small holes exist between allocated memory partitions;
  total memory space is available for a request, but it is not contiguous
- External fragmentation can be reduced by **compaction**
  - Relocate memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time
  - I/O problem
    - Cannot relocate process while it is involved in I/O
    - Do I/O only into OS buffers

(slide modified by R. Doemer, 05/18/10)