CS 343 - Operating Systems

Module-4A

Introduction to Memory Management

Dr. John Jose

Associate Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati

Overview of Memory Management

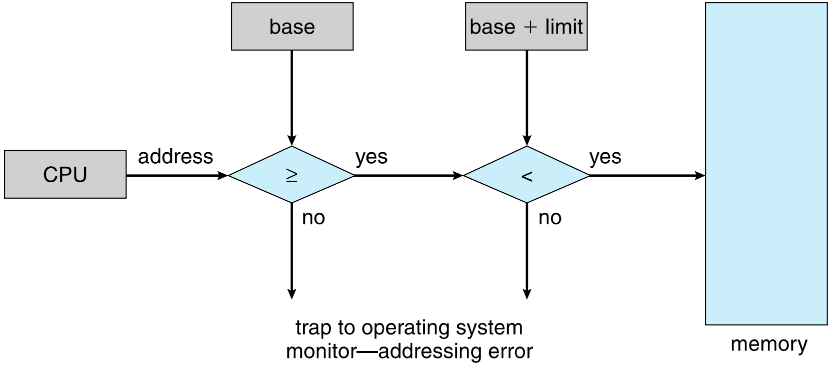
- Background
- Swapping
- Contiguous Memory Allocation
- Segmentation
- Paging
- Structure of the Page Table

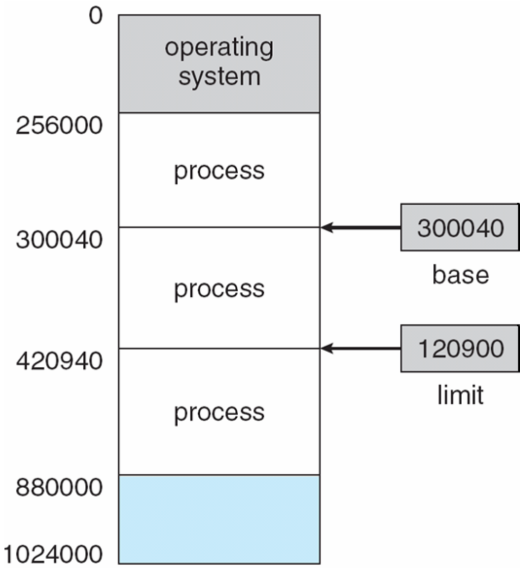
Background

- Program must be brought (from disk) into memory for the process to run
- CPU can access directly only main/cache memory and registers
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Register access in one CPU clock
- Main memory can take many cycles, causing a **stall**
- **Cache** is placed in between main memory and CPU registers
- Protection of memory required to ensure correct operation

Hardware Protection using Base and Limit Registers

- 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





Address Binding

- Programs on disk, ready to be brought into memory to execute.
- Without support, must be loaded into address 0000
- Inconvenient to have user process physical address always at 0000
- Addresses represented in different ways in a program’s life
  - Source code addresses usually symbolic
  - Compiled code addresses **bind** to relocatable addresses
    - i.e. “14 bytes from beginning of this module”
  - Linker or loader will bind relocatable addresses to absolute addresses
    - i.e. 74014

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
  - **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 (e.g., base and limit registers)

Logical vs. Physical Address Space

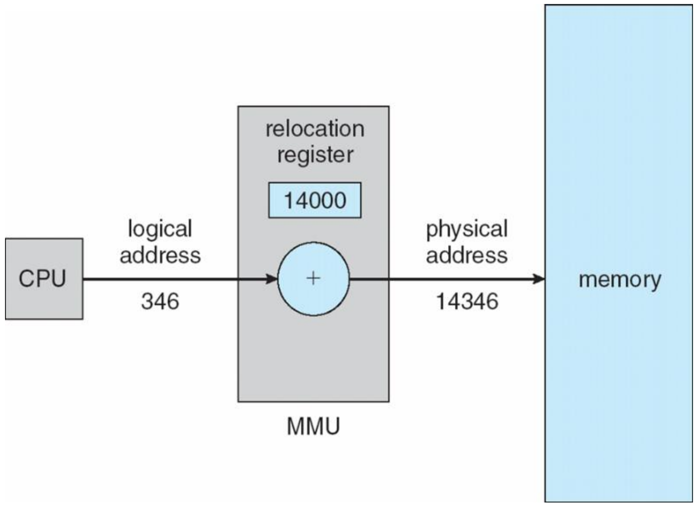
- **Logical address** – generated by the CPU; also referred to as **virtual address**
- **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time 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 of a system

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address at run time
- Simple relocation register based mapping
- 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

Dynamic relocation using a relocation register

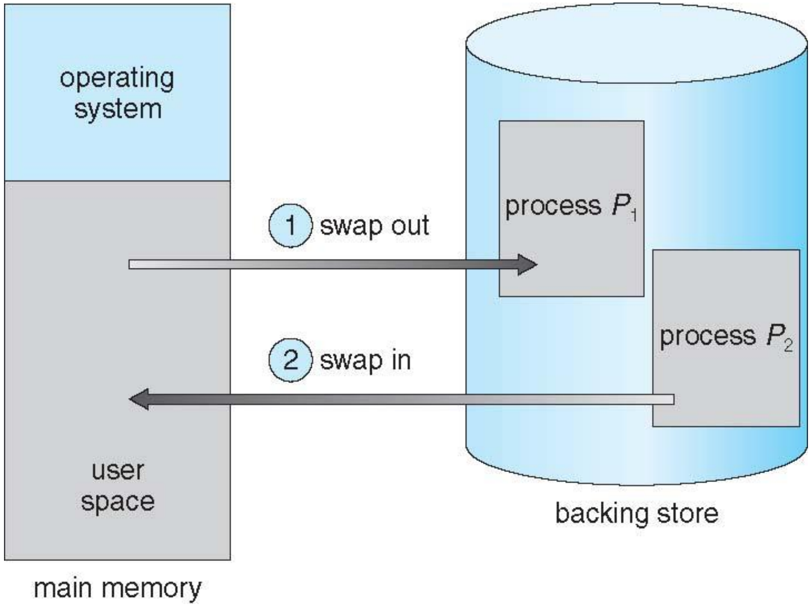
- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- Useful when large amounts of code are needed to handle infrequently occurring cases



Swapping

- A process can be **swapped** temporarily out of memory to a backing store, and then brought back into memory for continued execution
- Total virtual memory space of processes can exceed physical memory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Swapping



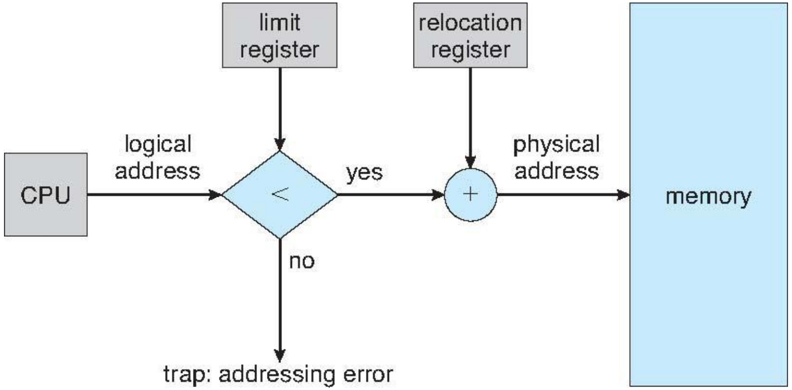
operating system
user space
main memory
1 swap out
2 swap in
process P_1
process P_2
backing store

Contiguous Allocation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory has usually into two **partitions**:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
  - Each process contained in single contiguous section of memory

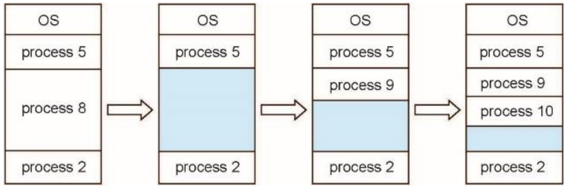
Contiguous Allocation

- Relocation registers 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 *dynamically*



Multiple-partition allocation

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency as per size of process
  - **Hole** – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    a) allocated partitions   b) free partitions (hole)



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; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the ***largest*** hole; must also search entire list
  - Produces the largest leftover hole
- First-fit and best-fit better than worst-fit in terms of speed and storage utilization

johnjose@iitg.ac.in

http://www.iitg.ac.in/johnjose/