Chapter 3 Topics:
Memory Management

Memory management Introduction

Main memory is a resource that must be allocated and deallocated. Memory Management Techniques determine:

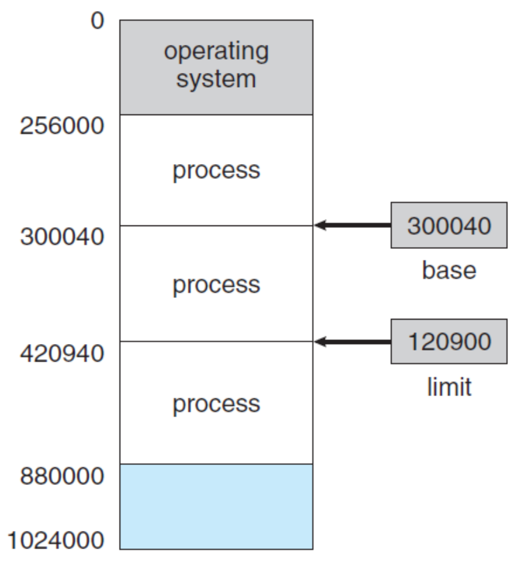
- How the memory is to be (logically) subdivided?
- Where and how a process resides in memory?
- How addressing is performed?
- How process can be relocated?
- How memory is to be protection?
- How memory can be shared by processes?
- How to logical and physically organize memory

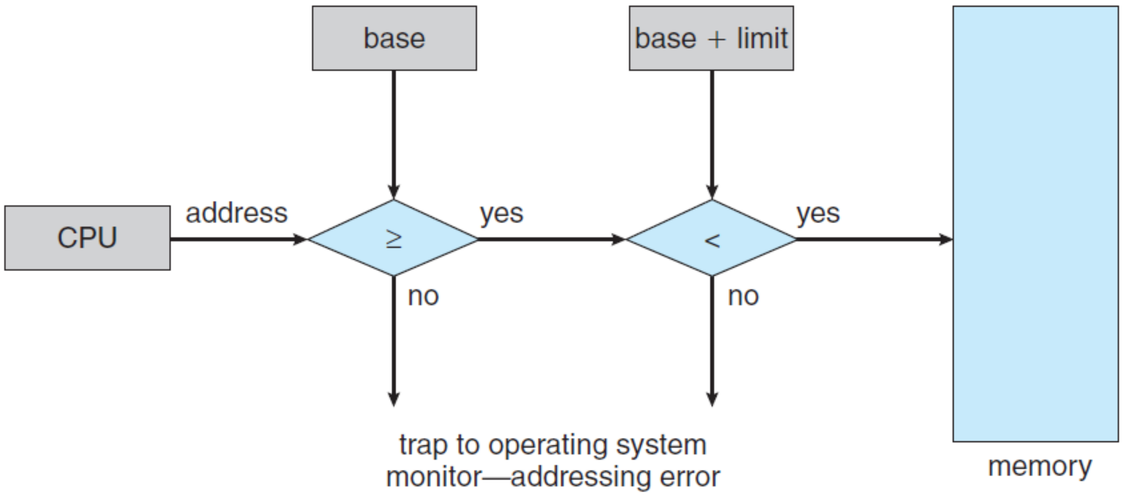
The memory manager is a component in Operating system used to manage memory. Managing and sharing of primary memory and minimizing the access time is the primary goal of memory manager.
Primary memory management functions include:

1. Allocate primary memory space to processes.
2. Minimize access time
3. Determining allocation policy for memory
4. Deallocation technique and policy.

Basic Hardware

- Each process has a separate memory space.
- The **base register** holds the smallest legal physical memory address; the **limit register** specifies the size of the range.
- Protection of memory space is accomplished by having the CPU hardware compare every address generated in user mode with the registers. Any attempt by a program executing in user mode to access operating-system memory or other users' memory results in a trap to the operating system, which treats the attempt as a **fatal** error This scheme prevents a user program from (accidentally or deliberately) modifying the code or data structures of either the operating system or other users.





Hardware protection for memory

Addressing Requirements of a Process

User programs go through several steps before being run

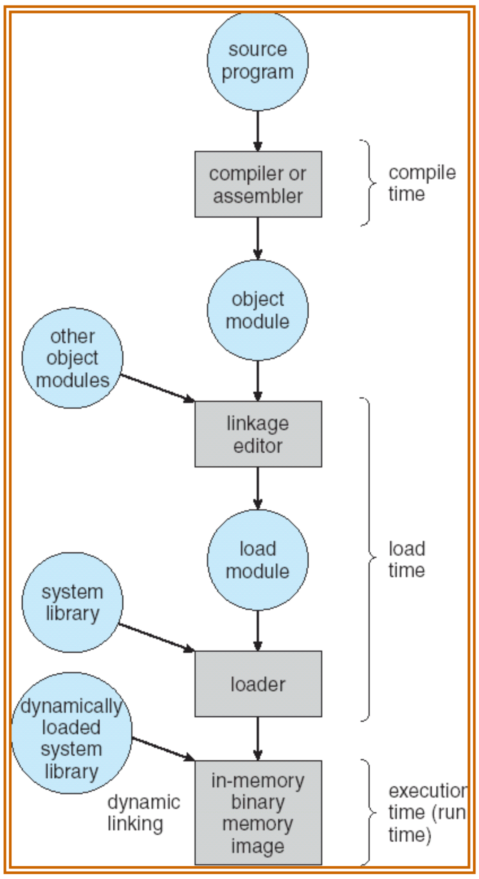
- source program - Addresses are generally symbolic (variable name).
- compiler time - Addresses are in relocatable format (in terms of offsets).
- linkage editor or loader - Addresses are in absolute addresses format (physical addresses).

Binding of Instructions and Data to Memory

To be executed, the program must be brought into memory and placed within a process.

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

- 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)

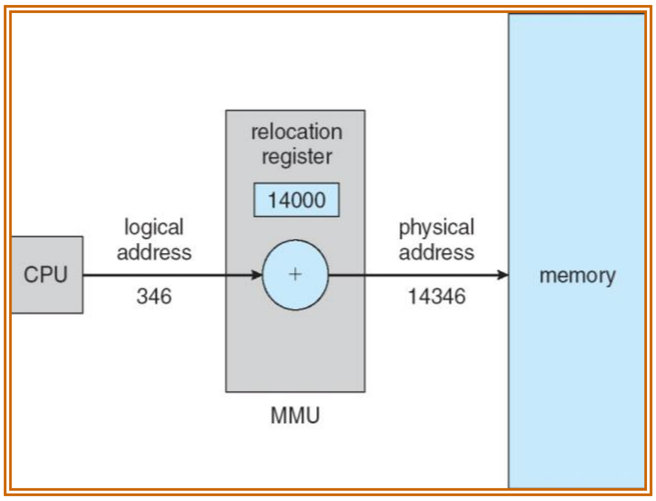
The memory management unit translates the logical address to physical address. It is a hardware device that maps virtual to physical address

In MMU scheme, the value in the 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.

Simple MMU scheme

- Relocation register contains the starting address of the program
- The value in the relocation register is added to every logical address generated by the CPU
- Dynamic binding can be implemented using a relocation register



Dynamic loading :

With dynamic loading, a routine is not loaded until it is called. All routines are kept on disk in a relocatable load format. The main program is loaded into memory and is executed.

Methods to execute programs larger than available memory

Overlays

Overlay is one of the techniques to run a program that is bigger than the size of the physical memory. The idea of overlays is to keep in memory only those instructions and data that are needed working.

- divide the program into modules in such a way that
- all modules are not needed in the memory at the same time.
- Programmer specifies which modules can overlay each other
- The linker inserts commands to invoke the loader
- When invoked linker replaces older modules with newer modules

Advantages

- Reduced memory requirements

Disadvantages

- Overlap map must be specified by programmer
- Programmer must know memory requirements

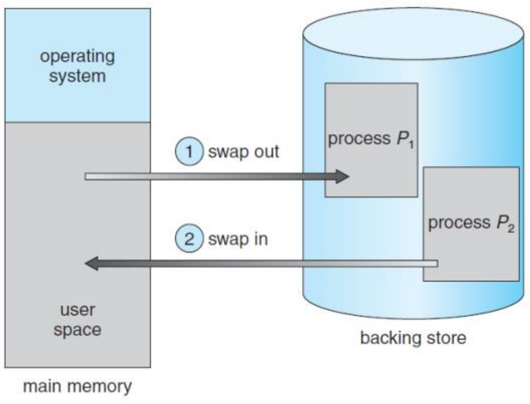
Swapping

A process needs to be in the memory to be executed. However, a process can be swapped temporarily out of memory to a backing store & then brought back into memory for continued execution.

Requirements for swap is a backing store – secondary storage disk. That it should be a fast disk ,large enough to accommodate copies of all memory images and provide direct access.

Steps involved

- Roll out :-Copying memory image of the process into backing store
- Roll in :-Copying memory image of the process from backing store to main memory



Uses

Used for implementing priority-based scheduling algorithms
o lower-priority process is swapped out
o higher-priority process can be loaded and executed.

Disadvantages

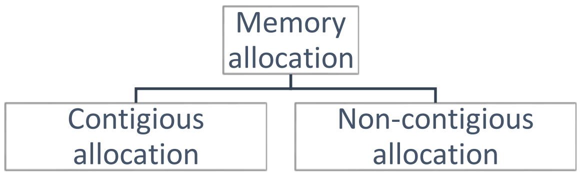
- Major part of swap time is transfer time
  - copying from memory to backing store
  - copying from backing store to memory
- Total transfer time is directly proportional to the amount of memory swapped.
- Context switching time is quite high in this scheme.
- Before swapping out a process, we have to ensure that it is completely idle.

Allocating Main Memory for programs

The operating system has to allocate memory for both user programs & System programs

Memory Allocation Techniques

The operating system follows different methods for allocating memory space for executing programs



Contiguous memory allocation

- Main memory usually divided into two partitions:
- Partition for Resident operating system
- Partition for User processes
- Every process is loaded into a single contiguous partition of the memory

Non-Contiguous memory allocation

- Main memory usually divided into multiple partitions:
- Process also can be split into multiple parts
- Each part of the program can be loaded into different parts of
- the memory
- Entire program need not be contiguous

Contiguous memory allocation

Types of Contiguous memory allocation

- Single-partition Allocation
- Multiple-Partition Allocation – Fixed Partitions
- Multiple-partition Allocation – Dynamic Partitions

Single-partition Allocation

- Memory split into two partitions one for Operating system and the other one for user programs. The user programs exist as one single partition

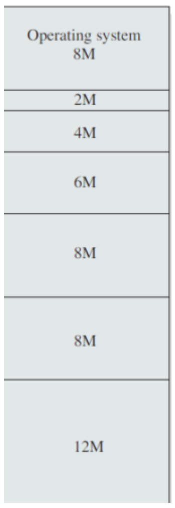
- A single process in loaded into the memory at a time. Next program can be loaded only after finishing current process

Advantages:

- Simplicity and no special hardware required

Disadvantages:

- Cannot support multiprogramming & multi user
- Cannot support multi-user.



Multiple-Partition Allocation – Fixed Partitions

Desirable in multiprogramming environment.

- Main memory is divided into number of static partitions at system generation time. A process may be loaded in to partition of equal or greater size. Partition size may be either equal or unequal as shown in the figure.
- The degree of multiprogramming is bounded by the number of
- Originally used in IBM OS/360 and no longer in use now.

Advantages

Simple to implement and little overhead

Disadvantage

Inefficient use of memory due to internal fragmentation; maximum number of active process is fixed.

Multiple-Partition Allocation – Dynamic Partitions

Partitions are of variable length and variable numbers. Process is allocated exactly the memory as required.

Working of dynamic partition scheme

Operating system maintains information about: Allocated partitions and Free partitions (hole).

Initially, all the memory is available and is considered as a single block (one big hole).

When a process arrives, we search for a hole large enough for the process

- If a hole of required size is found, it is allocated to the process
- If we find a hole, which is too large, it is split into two one part is allocated to the arriving process and the other is returned to the set of holes.
- When a process completes, the memory is freed and is placed in to the set of holes
- If the new hole is adjacent to other holes they are packed & merged into a single hole

Dynamic Allocation Placement Algorithms

How to satisfy a request of size n bytes from a list of free holes? Solutions to the problem of allocating memory

First-fit:

- Allocate the first hole that is big enough
- The search can start at the beginning
- or start from where the previous first-fit search was ended.

Best-fit:

- Allocate the smallest hole that is big enough
- Must search entire list
- Produces the smallest leftover hole.

Worst-fit:

- Allocate the largest available hole
- Must also search entire list

```
Produces Given five memory partitions of 100Kb, 500Kb, 200Kb, 300Kb,
600Kb (in order), how would the first-fit, best-fit, and worst-fit
algorithms place
   processes of 212 Kb, 417 Kb, 112 Kb, and 426 Kb (in order)? Which
algorithm makes the most efficient use of memory?

 First-fit:
 212K is put in 500K partition
 417K is put in 600K partition
 112K is put in 288K partition (new partition 288K = 500K - 212K)
 426K must wait

 Best-fit:
 212K is put in 300K partition
 417K is put in 500K partition
 112K is put in 200K partition
 426K is put in 600K partition

 Worst-fit:
 212K is put in 600K partition
 417K is put in 500K partition
 112K is put in 388K partition
 426K must wait
```

Internal Fragmentation

- Allocated memory may be slightly larger than requested memory.
- This size difference is internal to a memory partition
- Hence cannot be used by any other process.

External Fragmentation

- Total memory space exists to satisfy a request, but it is not contiguous.
- Hence cannot be used by any process.

External fragmentation can be reduced by compaction.

Compaction/De-fragmentation

Place all free memory together in one large block and all the allocated blocks together one side.