Memory Management

Subdividing memory to accommodate multiple processes

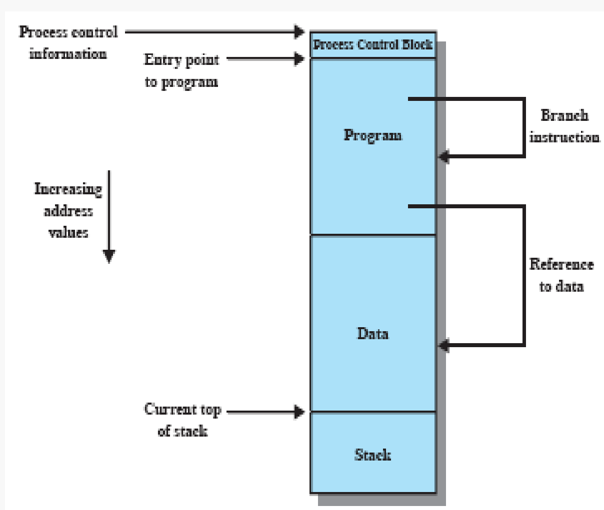
Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time

Relocation

- Programmer does not know where the program will be placed in memory when it is executed
- While the program is executing, it may be swapped to disk and returned to main memory at a different location (relocated)
- Memory references must be translated in the code to actual physical memory address

©2005 McQuain WD

Addressing Requirements



©2005 McQuain WD

Memory Management Requirements

Protection

- Processes should not be able to reference memory locations in another process without permission
- Impossible to check absolute addresses at compile time
- Must be checked at rum time
- Memory protection requirement must be satisfied by the processor (hardware) rather than the operating system (software)
  - Operating system cannot anticipate all of the memory references a program will make

Sharing

- Allow several processes to access the same portion of memory
- Better to allow each process access to the same copy of the program rather than have their own separate copy

©2005 McQuain WD

Memory Management Requirements

Logical Organization

- Programs are written in modules
- Modules can be written and compiled independently
- Different degrees of protection given to modules (read-only, execute-only)
- Share modules among processes

Physical Organization

- Memory available for a program plus its data may be insufficient
  - Overlaying allows various modules to be assigned the same region of memory
- Programmer does not know how much space will be available

©2005 McQuain WD

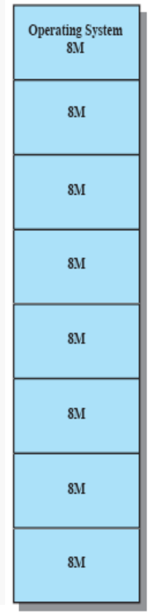
Fixed Partitioning

Equal-size partitions

- Any process whose size is less than or equal to the partition size can be loaded into an available partition
- If all partitions are full, the operating system can swap a process out of a partition
- A program may not fit in a partition. The programmer must design the program with overlays

Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This is called internal fragmentation.

Because all partitions are of equal size, it does not matter which partition is used, so placement algorithm is essentially trivial.



©2005 McQuain WD

Fixed Partitioning

Unequal-size partitions

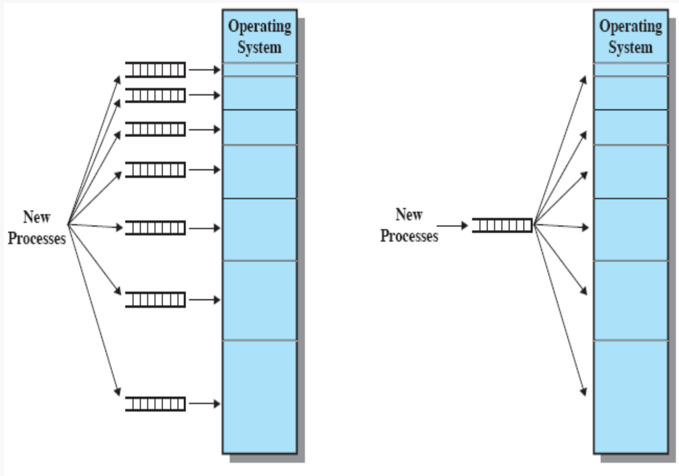
- Can assign each process to the smallest partition within which it will fit
- Queue for each partition
- Processes are assigned in such a way as to minimize wasted memory within a partition

Main memory use is potentially more efficient. Any program, no matter how small or large is placed in a closer-sized partition. Does not eliminate internal fragmentation.

Management of partitions is more complex, hence more overhead.

©2005 McQuain WD

Process Scheduling



(a) One process queue per partition

(b) Single queue

©2005 McQuain WD

Dynamic Partitioning

Partitions are of variable length and number

Process is allocated exactly as much memory as required

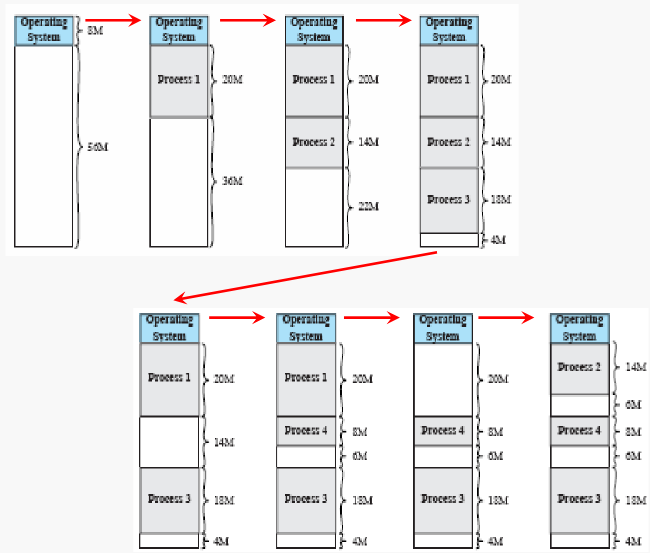
Eventually get holes in the memory. This is called external fragmentation

Must use *compaction* to shift processes so they are contiguous and all free memory is in one block

However, compaction is time-consuming, and it requires some scheme for adjusting addresses when relocating processes in memory

©2005 McQuain WD

Effect of Dynamic Partitioning



DP Placement Algorithm

Operating system must decide which free block to allocate to a process

Best-fit algorithm

- Chooses the block that is closest in size to the request
- Worst performer overall
- Since smallest block is found for process, the smallest amount of fragmentation is left
- Memory compaction must be done more often

First-fit algorithm

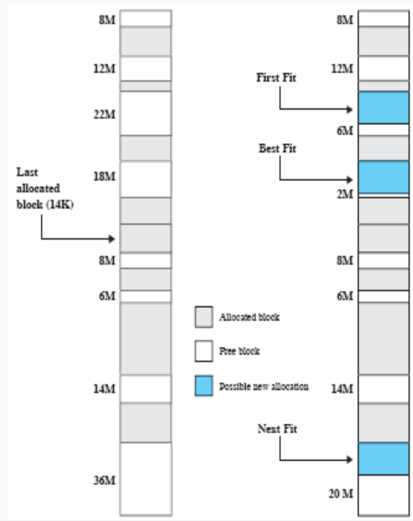
- Scans memory form the beginning and chooses the first available block that is large enough
- Fastest
- May have many process loaded in the front end of memory that must be searched over when trying to find a free block

Next-fit

- Scans memory from the location of the last placement
- More often allocate a block of memory at the end of memory where the largest block is found
- The largest block of memory is broken up into smaller blocks
- Compaction is required to obtain a large block at the end of memory

Examples

Before and after allocation of a 16MB block:

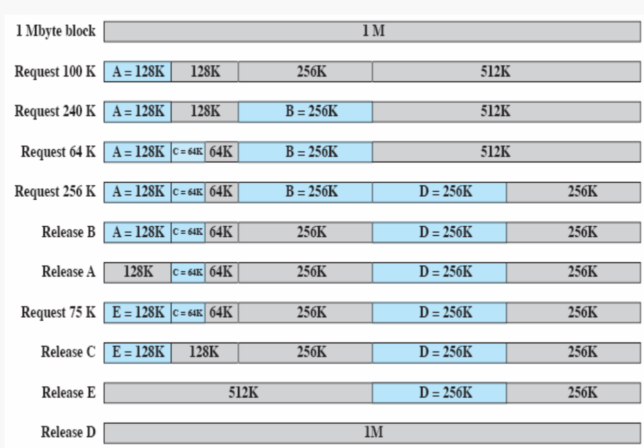


Buddy System

Entire space available is treated as a single block of 2^U

If a request of size s such that $2^{U-1} < s <= 2^U$, entire block is allocated

- Otherwise block is split into two equal buddies
- Process continues until smallest block greater than or equal to s is generated



Paging

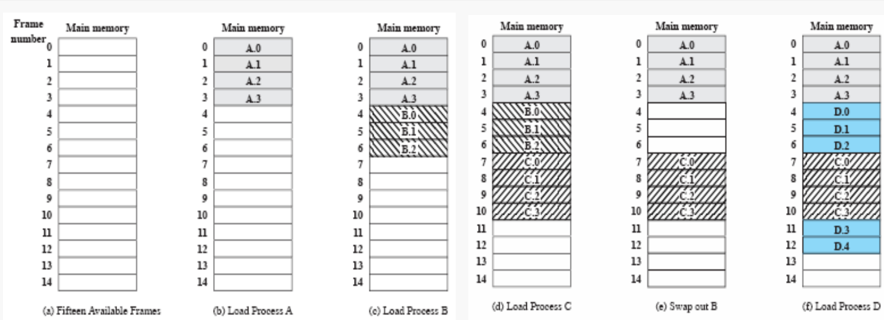
Partition memory into small equal fixed-size chunks and divide each process into the same size chunks

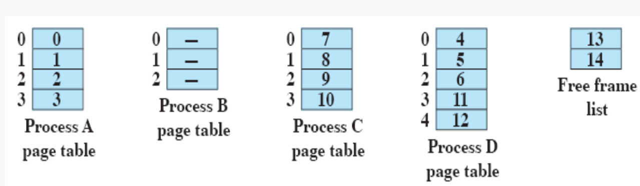
The chunks of a process are called pages and chunks of memory are called frames

Operating system maintains a page table for each process
- Contains the frame location for each page in the process
- Memory address consist of a page number and offset within the page

©2005 McQuain WD

Allocation of Free Frames





©2005 McQuain WD

Segmentation

All segments of all programs do not have to be of the same length

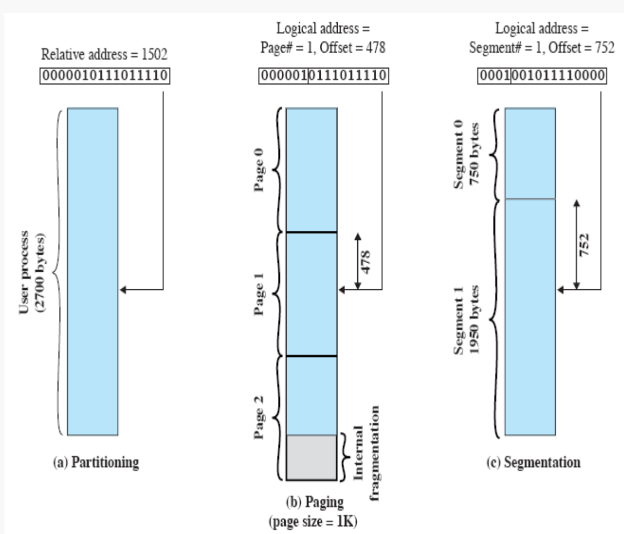
There is a maximum segment length

Addressing consist of two parts - a segment number and an offset

Since segments are not equal, segmentation is similar to dynamic partitioning

©2005 McQuain WD

Logical Addresses



©2005 McQuain WD

Address Translation

©2005 McQuain WD

Relocation and Addresses

When program loaded into memory the actual (absolute) memory locations are determined

A process may occupy different partitions at different times, which means different absolute memory locations during execution (from swapping)

Compaction will also cause a program to occupy a different partition which means different absolute memory locations

Paging and segmenting will fragment a process into discontiguous pieces.

Logical address
- Reference to a memory location independent of the current assignment of data to memory
- Translation must be made to the physical address

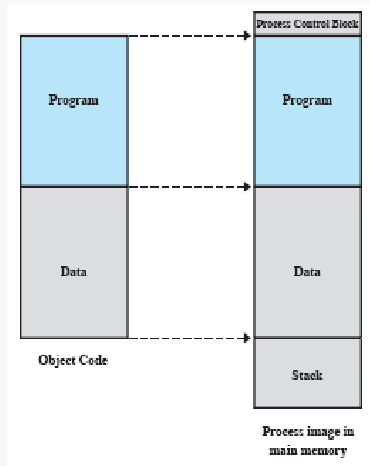
Relative address
- Address expressed as a location relative to some known point

Physical address
- The absolute address or actual location in main memory

©2005 McQuain WD

Process Image

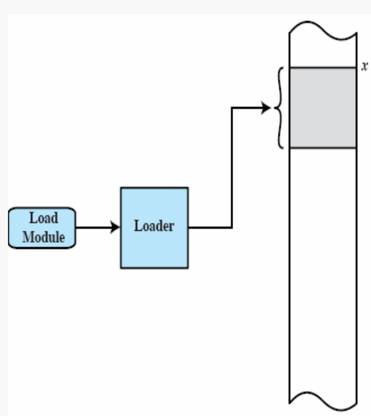
First step in process creation is to load the program into memory and create a process image.



Process Loading

The loader has the responsibility of placing the process image into memory, given the load module created by the linker.

Here's a typical overview (assuming contiguous allocation of main memory):

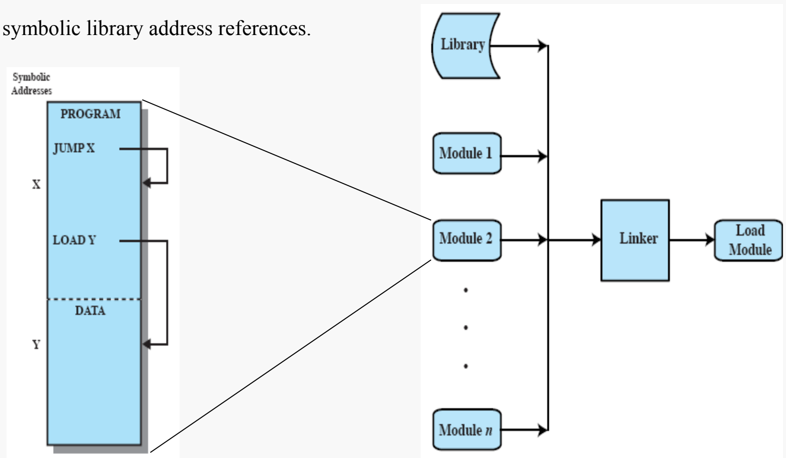


Linking

The linker takes a collection of object modules as input and produces a load module, consisting of an integrated set of program and data modules, which will be processed by the loader.

Resolves symbolic inter-module address references.

Resolves symbolic library address references.



©2005 McQuain WD

Absolute Load Module

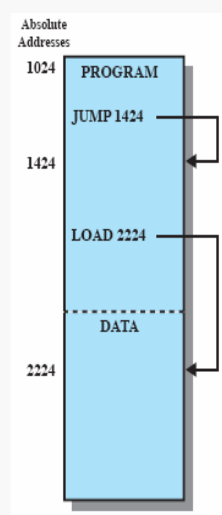
An absolute load module would be created by mapping references to absolute addresses depending upon a specific base address at load time:

Link-time binding of references to physical addresses.

Simple processing by linker.

Loader simply copies load module into memory.

Loader MUST copy load module to correct base address or program will not execute correctly.



©2005 McQuain WD

Relative Load Module

A relative load module would be created by mapping references to relative addresses that would be translated to physical addresses at load time.

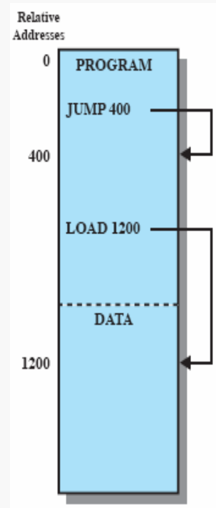
Load-time binding of relative references to physical addresses.

Linker must leave tags in the load module to guide the loader's actions – *relocation dictionary*.

Loader must find each relative address in the load module and add the base address (determined at load time) to it.

Too expensive for systems that use swapping to control system load.

Logically inadequate for systems that use paging or segmented memory management.



©2005 McQuain WD

Dynamically Relocatable Load Module

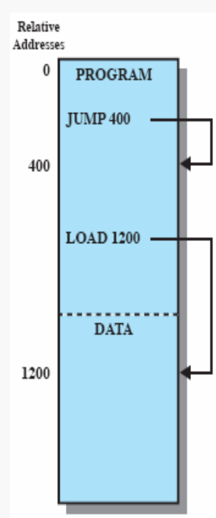
A dynamically relocatable load module would also be created by mapping references to relative addresses that would be translated to physical addresses at run time.

Run-time binding of relative references to physical addresses.

Loader merely copies load module with relative addresses into memory.

Hardware support MUST be provided to make this efficient.

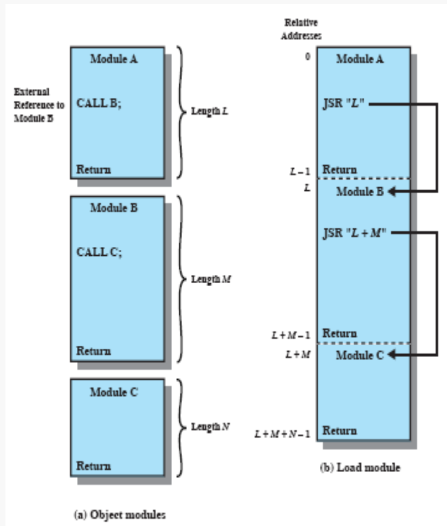
Dominant form of loading on modern systems.



©2005 McQuain WD

Object Module to Load Module

The linker that produces a relocatable load module works something like this:



(b) Load module

(a) Object modules

©2005 McQuain WD

Relocation and Hardware

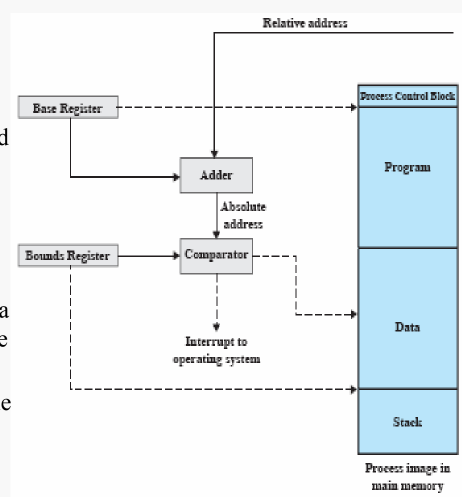
Base register

Starting address for the process

Bounds register

Ending location of the process

These values are set when the process is loaded or when the process is swapped in



The value of the base register is added to a relative address to produce an absolute address

The resulting address is compared with the value in the bounds register

If the address is not within bounds, an interrupt is generated to the operating system

©2005 McQuain WD