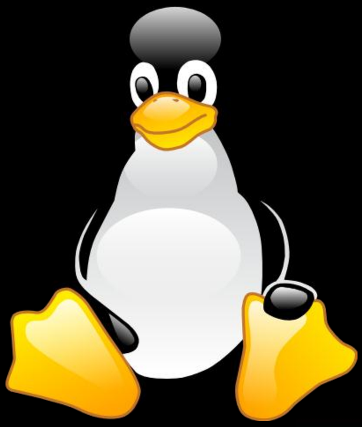
Tushar B Kute,
Assistant Professor,
Sandip Institute of Technology and Research Centre, Nashik (INDIA)
tbkute@gmail.com

University of Pune
T.E. I.T.
Subject code: 314441

OPERATING SYSTEM

Part 19: Memory Management Basics

Unit- IV



Detailed Syllabus

- Unit IV Memory Management 9 Hrs.
  - Memory Management requirements, Memory partitioning: Fixed, Dynamic partitioning, Buddy System.
  - Memory allocation Strategies (First Fit, Best Fit, Worst Fit, Next Fit), Fragmentation, Swapping, Segmentation.
  - Paging, Virtual Memory, Demand paging
  - Page Replacement Policies (FIFO, LRU, Optimal, Clock), Thrashing, Working Set Model.

Roadmap

- Basic requirements of Memory Management
- Memory Partitioning
- Basic blocks of memory management
  - Paging
  - Segmentation

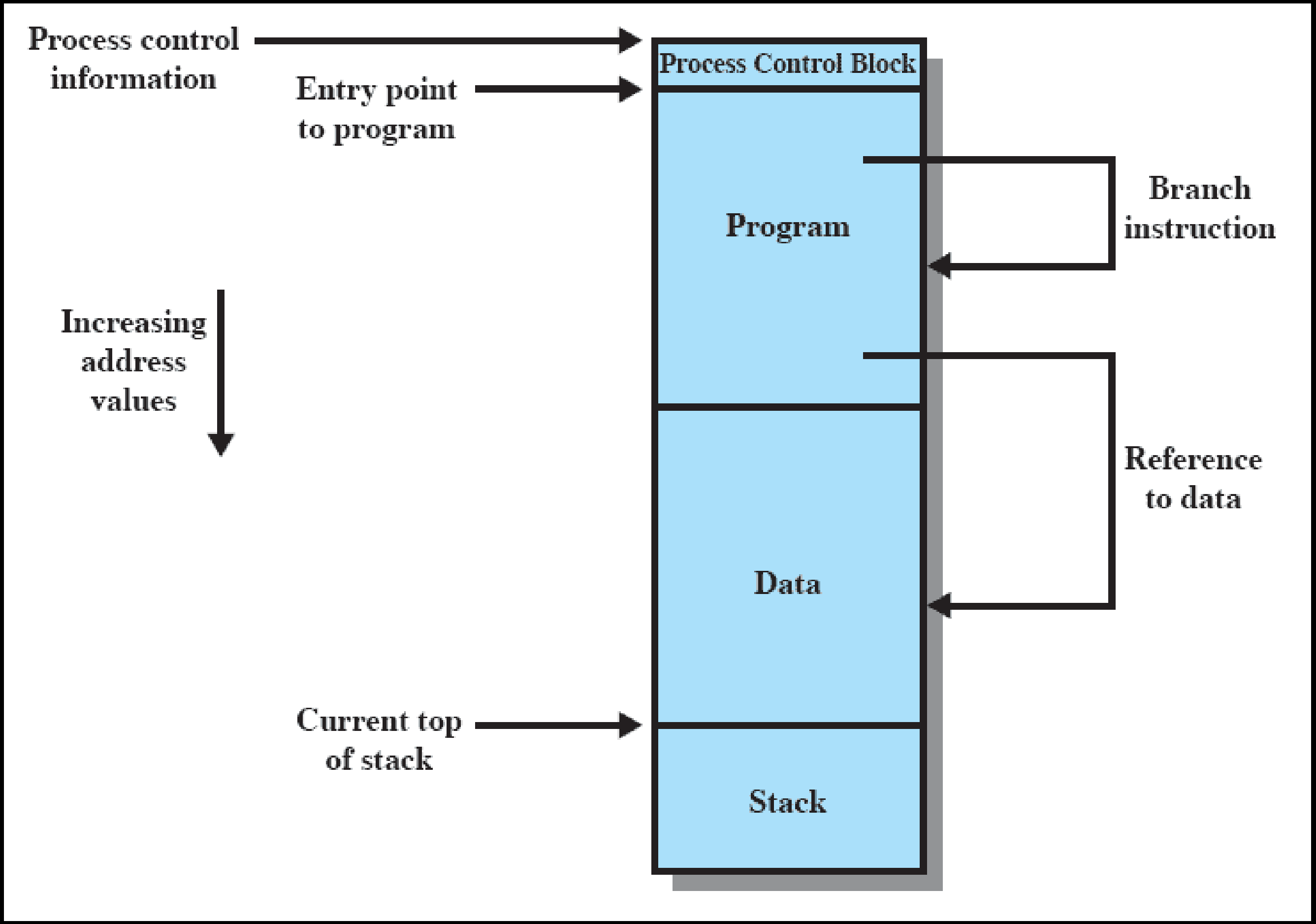
The need for memory management

- Memory is cheap today, and getting cheaper
  - But applications are demanding more and more memory, there is never enough!
- Memory Management, involves swapping blocks of data from secondary storage.
- Memory I/O is slow compared to a CPU
  - The OS must cleverly time the swapping to maximise the CPU's efficiency

Memory Management

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

Addressing



Process control information
Entry point to program
Process Control Block
Program
Branch instruction
Increasing address values
Reference to data
Data
Current top of stack
Stack

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 run time

Requirements : 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

Requirements: Logical Organization

- Memory is organized linearly (usually)
- 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
- Segmentation helps here

Requirements: Physical Organization

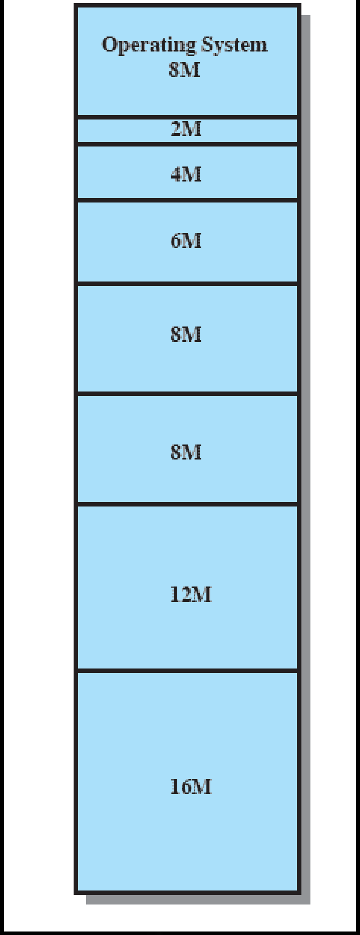
- Cannot leave the programmer with the responsibility to manage memory
- Memory available for a program plus its data may be insufficient
  - Overlaying allows various modules to be assigned the same region of memory but is time consuming to program
- Programmer does not know how much space will be available

Partitioning

- An early method of managing memory
  - Pre-virtual memory
  - Not used much now
- But, it will clarify the later discussion of virtual memory if we look first at partitioning
  - Virtual Memory has evolved from the partitioning methods

Solution – Unequal Size Partitions

- Lessens both problems
  - but doesn’t solve completely
- In Fig.,
  - Programs up to 16M can be accommodated without overlay
  - Smaller programs can be placed in smaller partitions, reducing internal fragmentation



Placement Algorithm

- Equal-size
  - Placement is trivial (no options)
- Unequal-size
  - 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

Fixed Partitioning

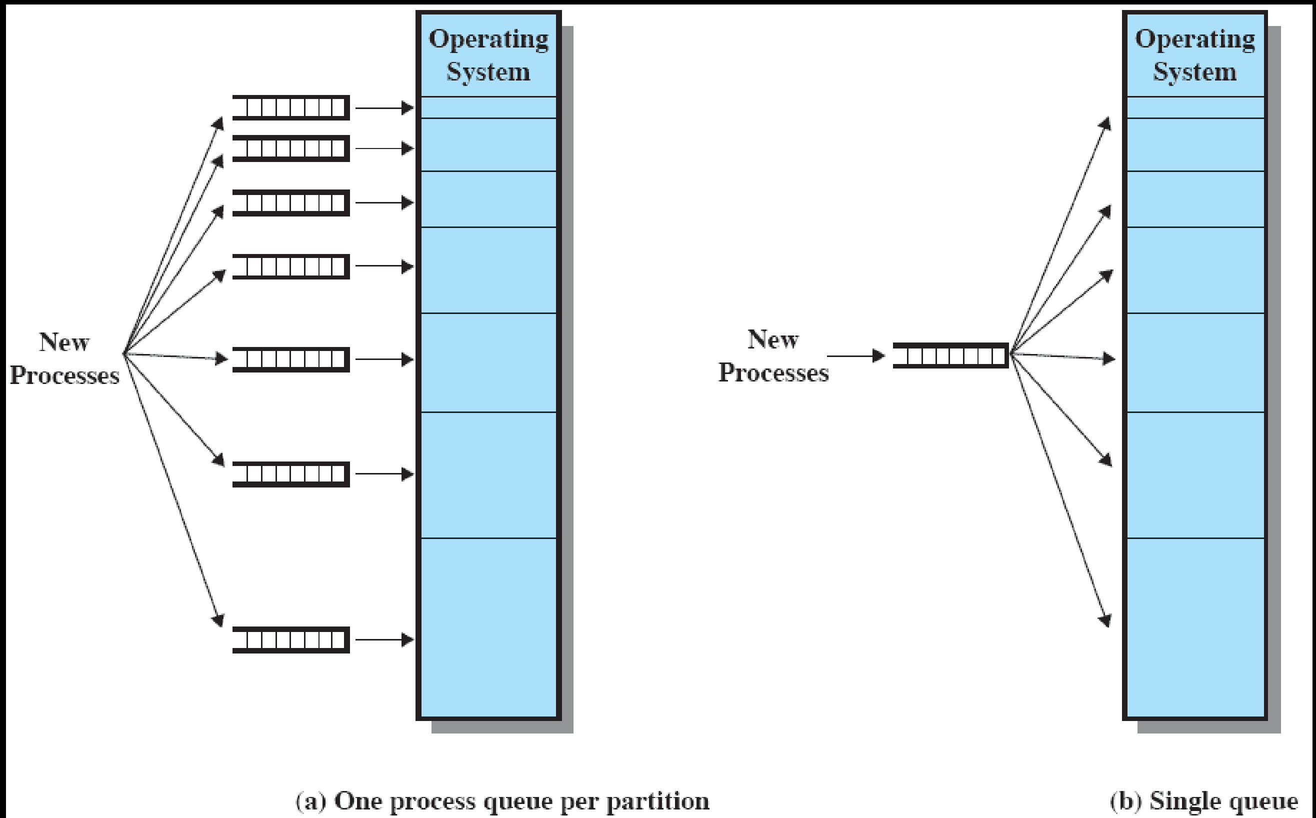


Figure 7.3 Memory Assignment for Fixed Partitioning

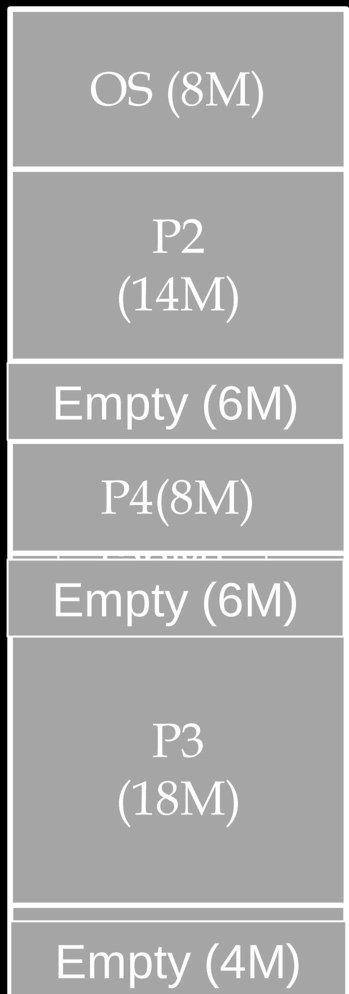
Remaining Problems with Fixed Partitions

- The number of active processes is limited by the system
  - i.e. limited by the pre-determined number of partitions
- A large number of very small process will not use the space efficiently
  - In either fixed or variable length partition methods

Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required

Dynamic Partitioning Example



- External Fragmentation
- Memory external to all processes is fragmented
- Can resolve using compaction
  - OS moves processes so that they are contiguous
  - Time consuming and wastes CPU time

Dynamic Partitioning

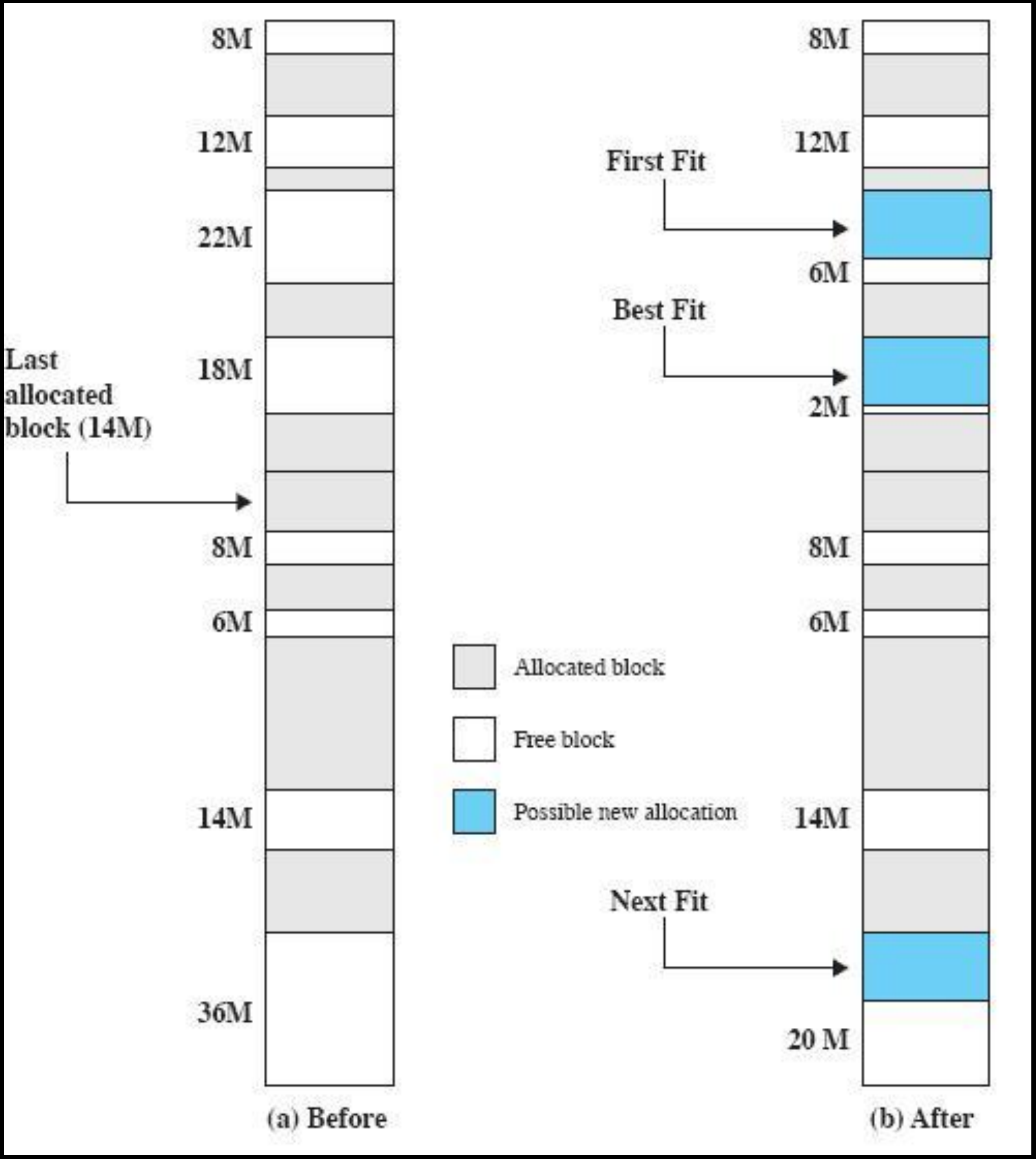
- Operating system must decide which free block to allocate to a process
- Best-fit algorithm
  - Chooses the block that is closest in size to the request
  - Worst performer overall
  - Since smallest block is found for process, the smallest amount of fragmentation is left
  - Memory compaction must be done more often

Dynamic Partitioning

- 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

Dynamic Partitioning

- 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


8M
12M
22M
18M
Last allocated block (14M)
8M
6M
14M
36M
(a) Before
Allocated block
Free block
Possible new allocation
First Fit
Best Fit
Next Fit
8M
12M
6M
2M
8M
6M
14M
20 M
(b) After

Allocation

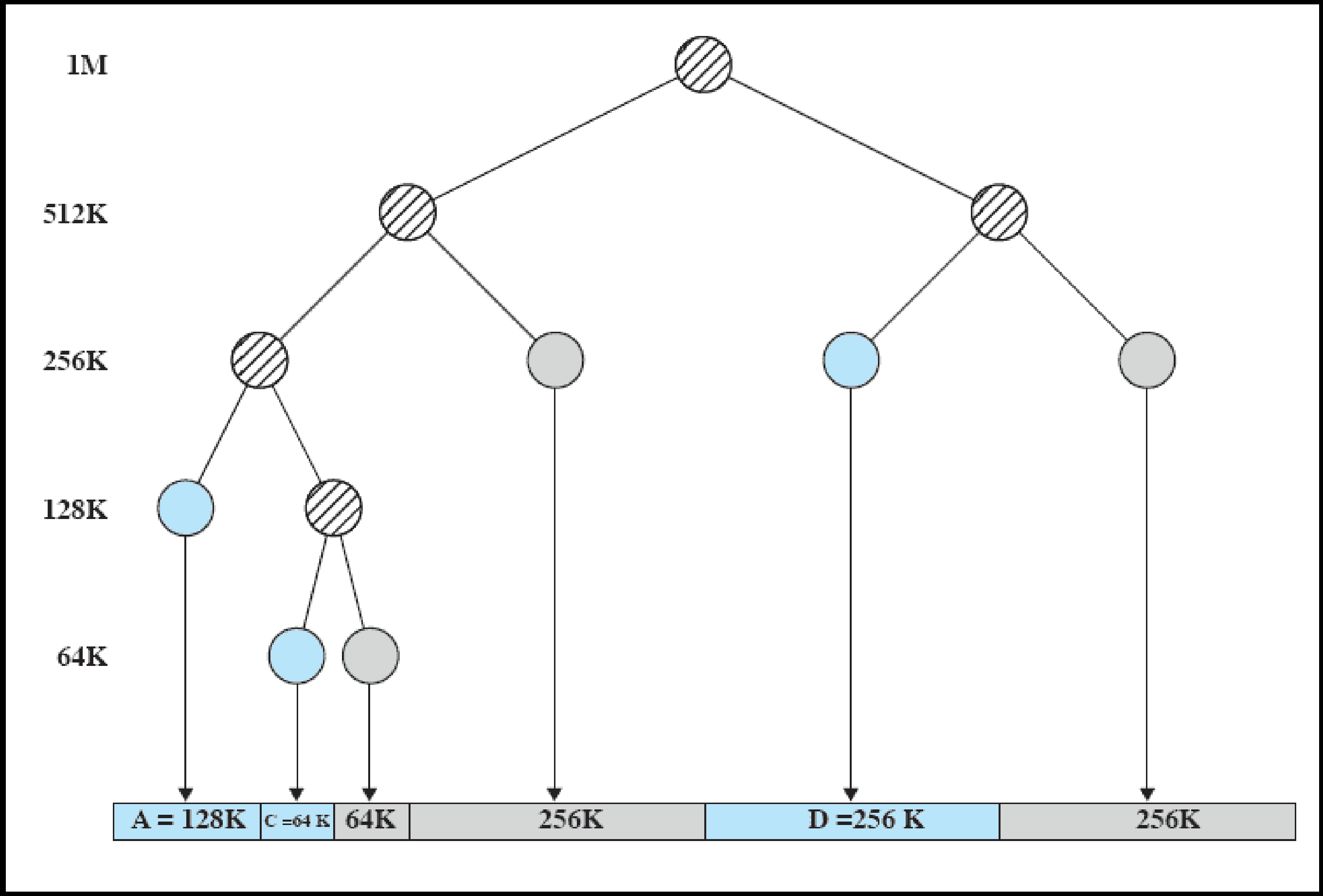
Buddy System

- Entire space available is treated as a single block of 2U
- If a request of size s where 2U-1 < s <= 2U
  - 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

Example of Buddy System

1 Mbyte block	1 M					
Request 100 K	A = 128K	128K	256K	512K		
Request 240 K	A = 128K	128K	B = 256K	512K		
Request 64 K	A = 128K	C = 64K	64K	B = 256K	512K	
Request 256 K	A = 128K	C = 64K	64K	B = 256K	D = 256K	256K
Release B	A = 128K	C = 64K	64K	256K	D = 256K	256K
Release A	128K	C = 64K	64K	256K	D = 256K	256K
Request 75 K	E = 128K	C = 64K	64K	256K	D = 256K	256K
Release C	E = 128K	128K	256K	D = 256K	256K	
Release E	512K	D = 256K	256K			
Release D	1M					

Tree Representation of Buddy System



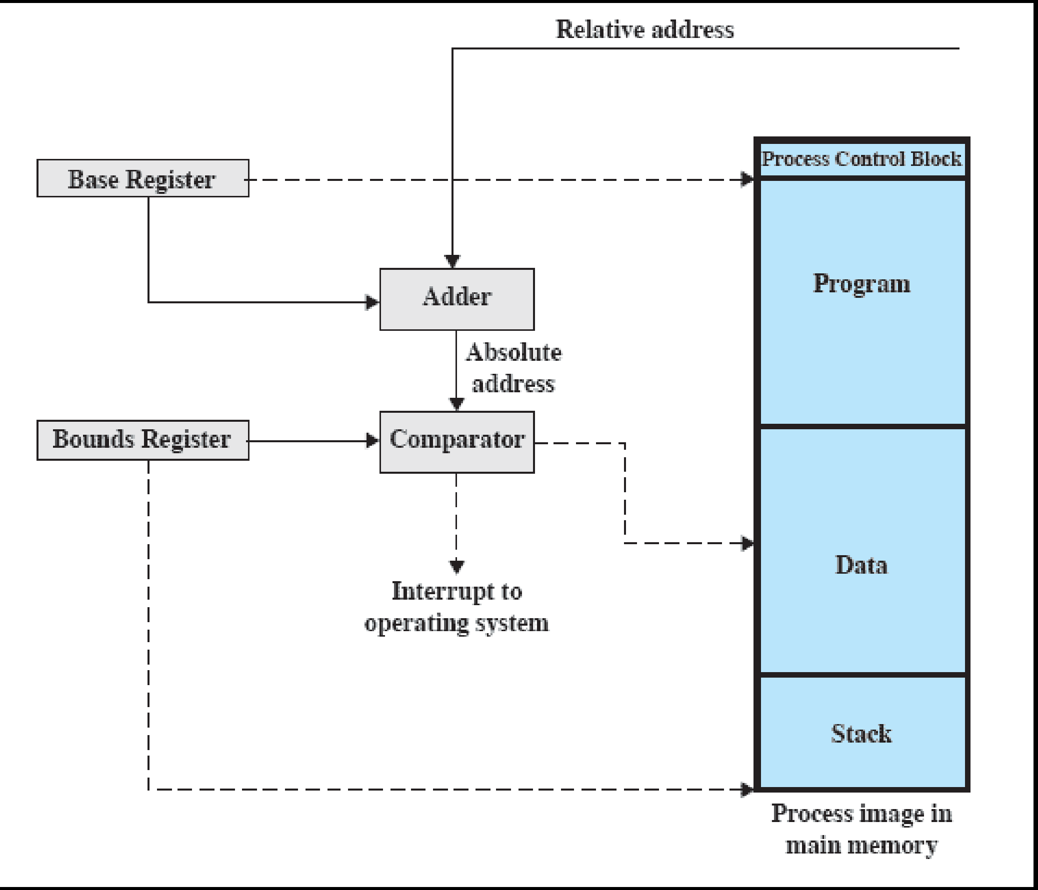
Relocation

- When program loaded into memory the actual (absolute) memory locations are determined
- A process may occupy different partitions which means different absolute memory locations during execution
  - Swapping
  - Compaction

Addresses

- Logical
  - Reference to a memory location independent of the current assignment of data to memory.
- Relative
  - Address expressed as a location relative to some known point.
- Physical or Absolute
  - The absolute address or actual location in main memory.

Relocation


Relative address
Base Register
Process Control Block
Program
Adder
Absolute address
Bounds Register
Comparator
Data
Interrupt to operating system
Stack
Process image in main memory

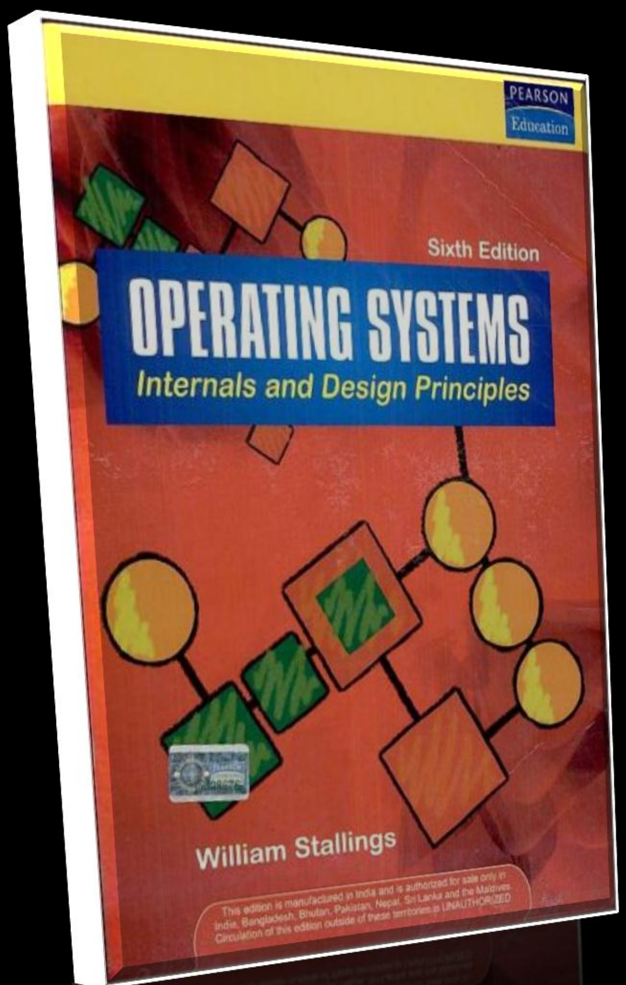
Registers Used during Execution

- 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

Registers Used during Execution

- 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

Reference Books



- “Operating System: Internals and Design Principles” by *William Stallings*, Pearson Education.