Understanding Operating Systems Sixth Edition

Chapter 2
Memory Management: Early Systems

Learning Objectives

After completing this chapter, you should be able to describe:

- The basic functionality of the three memory allocation schemes presented in this chapter: fixed partitions, dynamic partitions, relocatable dynamic partitions
- Best-fit memory allocation as well as first-fit memory allocation schemes
- How a memory list keeps track of available memory

Learning Objectives (cont'd.)

- The importance of deallocation of memory in a dynamic partition system
- The importance of the bounds register in memory allocation schemes
- The role of compaction and how it improves memory allocation efficiency

Introduction

- Management of main memory is critical
- Entire system performance dependent on two items
  - How much memory is available
  - Optimization of memory during job processing
- This chapter introduces:
  - Memory manager
  - Four types of memory allocation schemes
    - Single-user systems
    - Fixed partitions
    - Dynamic partitions
    - Relocatable dynamic partitions

Introduction

- Five steps in development cycle
  - Write code
  - Use translator to translate
  - Link modules
  - Load into memory
  - Run the program
- Early developers
  - Assigned the addresses manually
  - Wrote programs in machine language
  - Entered them into the memory manually by manipulating switches, buttons on the front panel machines

Dream of a programmer

- Private
- Infinitely Large
- Infinitely Fast
- Does not lose contents when no electricity
- Inexpensive

Alternative?

Memory hierarchy, together with memory manager

What goes into main memory

- The instructions and data required by the operating system to run
- Instructions and data for every open application
- Instructions and data for each users program
- Data related to calculations the system must perform

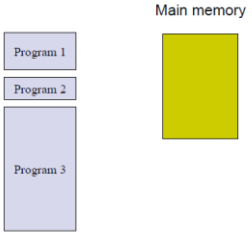
Single-User Contiguous Scheme

- Commercially available in 1940s and 1950s
- Entire program loaded into memory
- Contiguous memory space allocated as needed
- Jobs processed sequentially
- Memory manager performs minimal work
  1. Evaluates incoming process size: loads if small enough to fit; otherwise, rejects and evaluates next incoming process
  2. Monitors occupied memory space; when process ends, makes entire memory space available and returns to Step 1

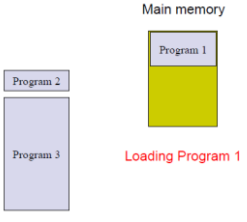
Single-User Contiguous Scheme (cont'd.)

- Disadvantages
  - No support for multiprogramming or networking
  - Not cost effective: unsuitable for business when introduced in late 1940s and early 1950s
  - Program size must be less than memory size to execute

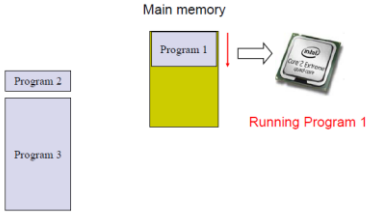
Single-User Contiguous Scheme (cont'd.)



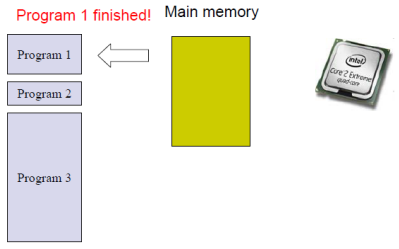
Single-User Contiguous Scheme (cont'd.)



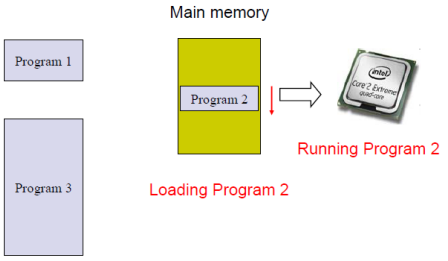
Single-User Contiguous Scheme (cont'd.)



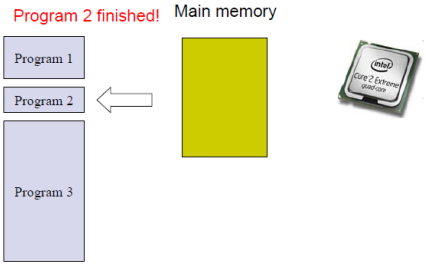
Single-User Contiguous Scheme (cont'd.)



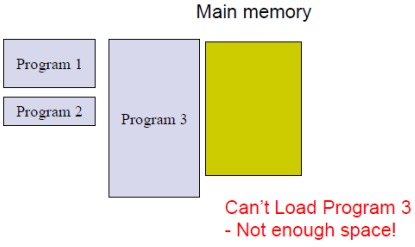
Single-User Contiguous Scheme (cont'd.)



Single-User Contiguous Scheme (cont'd.)



Single-User Contiguous Scheme (cont'd.)



Fixed Partitions

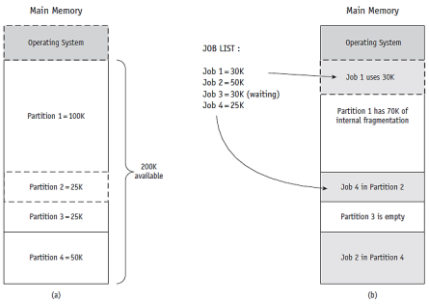
- Commercially available in 1950s and 1960s
- Main memory is partitioned
  - *At system startup*
  - One contiguous partition per job
- Permits multiprogramming
- Partition sizes remain static
  - Must shut down computer system to reconfigure
- Requires:
  - Protection of the job's memory space
  - Matching job size with partition size

Fixed Partitions (cont'd.)

- Memory manager allocates memory space to jobs
  - Uses a table

Partition Size	Memory Address	Access	Partition Status
100K	200K	Job 1	Busy
25K	300K	Job 4	Busy
25K	325K		Free
50K	350K	Job 2	Busy

(table 2.1)
A simplified fixed-partition memory table with the free partition shaded.



(figure 2.2)
As the jobs listed in Table 2.1 are loaded into the four fixed partitions, Job 3 must wait even though Partition 1 has 70K of available memory. Jobs are allocated space on the basis of "first available partition of required size."

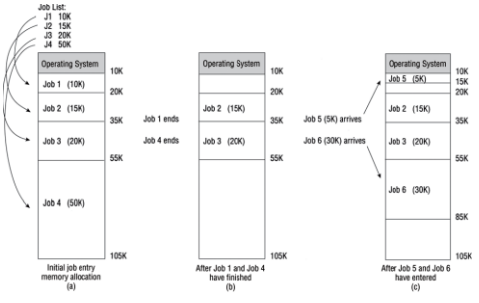
Fixed Partitions (cont'd.)

- Disadvantages
  - Requires contiguous loading of entire program
  - Job allocation method
    - First available partition with required size
  - To work well:
    - All jobs must be same size and memory size known ahead of time
  - Arbitrary partition size leads to undesired results
    - Partition too small
      - Large jobs have longer turnaround time
    - Partition too large
      - Memory waste: **internal fragmentation**

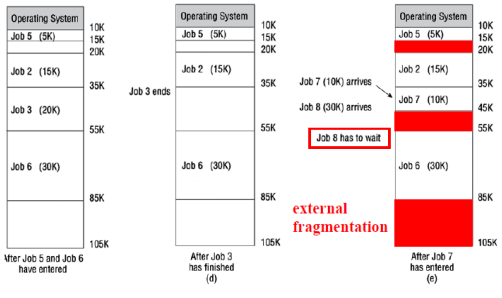
Dynamic Partitions

- Main memory is partitioned
  - Jobs given only as much as memory requested when *loaded*
  - One contiguous partition per job
- Job allocation method
  - **First come, first serve** allocation method
  - Memory waste: comparatively small
- Disadvantages
  - Full memory utilization only during loading of first jobs
  - Subsequent allocation: memory waste
    - External fragmentation: fragments between blocks

Dynamic Partitions (cont'd.)



Dynamic Partitions (cont'd.)



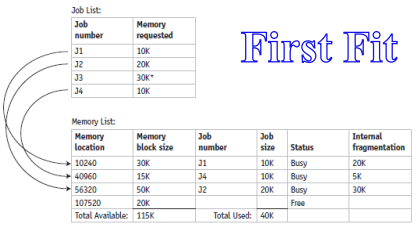
Best-Fit Versus First-Fit Allocation

- Two methods for free space allocation
  - **First-fit memory allocation:** first partition fitting the requirements
    - Leads to fast allocation of memory space
    - Memory Manager: free/busy lists organized by memory locations (low- to high-order memory)
  - **Best-fit memory allocation:** smallest partition fitting the requirements
  - Memory Manager: free/busy lists ordered by size (smallest to largest)
    - Results in least wasted space
    - Fragmentation reduced, but not eliminated
- Fixed and dynamic memory allocation schemes use both methods

Best-Fit Versus First-Fit Allocation (cont'd.)

- **First-fit memory allocation**
  - Advantage: faster in making allocation
  - Disadvantage: leads to memory waste
- **Best-fit memory allocation**
  - Advantage: makes the best use of memory space
  - Disadvantage: slower in making allocation

Best-Fit Versus First-Fit Allocation (cont'd. Fixed Partition)



(figure 2.5)
Using a first-fit scheme, Job 1 claims the first available space. Job 2 then claims the first partition large enough to accommodate it, but by doing so it takes the last block large enough to accommodate Job 3. Therefore, Job 3 (indicated by the asterisk) must wait until a large block becomes available, even though there's 75K of unused memory space (internal fragmentation).

Best-Fit Versus First-Fit Allocation (cont'd.)

(figure 2.6)
Best-fit free scheme. Job 1 is allocated to the closest-fitting free partition, as are Job 2 and Job 3. Job 4 is allocated to the only available partition although it isn't the best-fitting one. In this scheme, all four jobs are served without waiting. Notice that the memory list is ordered according to memory size. This scheme uses memory more efficiently but it's slower to implement.

Best-Fit Versus First-Fit Allocation (cont'd.)

- **Algorithm for first-fit**
  - Assumes memory manager keeps two lists
    - One for free memory
    - One for busy memory blocks
  - Compares the size of each job to the size of each free memory block
    - Until a block is found that is large enough to fit the job
  - Job stored into that block of memory
  - Memory Manager moves out of the loop
    - Fetches next job from the entry queue

Best-Fit Versus First-Fit Allocation (cont'd.)

- **Algorithm for first-fit (cont'd.):**
  - If entire list searched: no memory block large enough
    - Then job is placed into waiting queue
    - Memory Manager fetches next job
  - Process repeats

Best-Fit Versus First-Fit Allocation (cont'd.)

Status Before Request		Status After Request	
Beginning Address	Free Memory Block Size	Beginning Address	Free Memory Block Size
4075	105	4075	105
5225	5	5225	5
6785	600	*6985	400
7560	20	7560	20
7600	205	7600	205
10250	4050	10250	4050
15125	230	15125	230
24500	1000	24500	1000

(table 2.2)
These two snapshots of memory show the status of each memory block before and after 200 spaces are allocated at address 6785, using the first-fit algorithm. (Note: All values are in decimal notation unless otherwise indicated.)

Best-Fit Versus First-Fit Allocation (cont'd.)

- **Algorithm for best-fit**
  - Goal
    - Find the smallest memory block into which the job will fit

Best-Fit Versus First-Fit Allocation (cont'd.)

Status Before Request		Status After Request	
Beginning Address	Free Memory Block Size	Beginning Address	Free Memory Block Size
4075	105	4075	105
5225	5	5225	5
6785	600	6785	600
7560	20	7560	20
7600	205	*7800	5
10250	4050	10250	4050
15125	230	15125	230
24500	1000	24500	1000

(table 2.3)
These two snapshots of memory show the status of each memory block before and after 200 spaces are allocated at address 7600, using the best-fit algorithm.

Best-Fit Versus First-Fit Allocation (cont'd.)

- **Hypothetical allocation schemes**
  - Next-fit: starts searching from last allocated block, for next available block when a new job arrives
  - Worst-fit: allocates largest free available block to new job
    - Opposite of best-fit
    - Good way to explore theory of memory allocation
    - Not best choice for an actual system

Deallocation

- **Deallocation:** freeing allocated memory space
- For fixed-partition system:
  - Straightforward process
  - Memory Manager resets the status of job's memory block to "free" upon job completion
  - Any code may be used
  - Example code: binary values with zero indicating free and one indicating busy: e.g., 0 = free and 1 = busy

Deallocation (cont'd.)

- For dynamic-partition system:
  - Algorithm tries to combine free areas of memory
  - More complex
- Three dynamic partition system cases: depending on position of block to be deallocated
  - **Case 1:** adjacent to another free block
  - **Case 2:** between two free blocks
  - **Case 3:** isolated from other free blocks

Case 1: Joining Two adjacent Free Blocks

- Blocks are adjacent
- List changes to reflect starting address of the new free block
  - Example: 7600 - the address of the first instruction of the job that just released this block
- Memory block size changes to show its new size for the new free space
  - Combined total of the two free partitions
  - Example: (200 + 5)

Case 1: Joining Two adjacent Free Blocks

(table 2.4)
This is the original free list before deallocation for Case 1. The asterisk indicates the free memory block (of size 5) that's adjacent to the soon-to-be-free memory block (of size 200) that's shaded.
© Cengage Learning 2014

Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	20	Free
(7600)	(200)	(Busy)[1]
*7800	5	Free
10250	4050	Free
15125	230	Free
24500	1000	Free

[1]Although the numbers in parentheses don't appear in the free list, they've been inserted here for clarity. The job size is 200 and its beginning location is 7600.

Case 1: Joining Two Free Blocks (cont'd.)

Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	20	Free
*7600	205	Free
10250	4050	Free
15125	230	Free
24500	1000	Free

(table 2.5)
Case 1. This is the free list after deallocation. The shading indicates the location where changes were made to indicate the free memory block (of size 205).
© Cengage Learning 2014

Case 2: Joining Three Free Blocks

- Deallocated memory space
  - Between two free memory blocks
- List changes: reflect starting address of new free block
  - Example: smallest beginning address (7560)
- Three free partitions' sizes: combined
  - Example: (20 + 20 + 205)
- Total size: stored with smallest beginning address
- Last partition: assigned null entry status

Case 2: Joining Three Free Blocks (cont'd.)

Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
*7560	20	Free
(7580)	(20)	(Busy)[1]
*7600	205	Free
10250	4050	Free
15125	230	Free
24500	1000	Free

[1] Although the numbers in parentheses don't appear in the free list, they have been inserted here for clarity.

(table 2.6)
Case 2. This is the Free List before deallocation. The asterisks indicate the two free memory blocks that are adjacent to the soon-to-be-free memory block.
© Cengage Learning 2014

Case 2: Joining Three Free Blocks

- Deallocated memory space
  - Between two free memory blocks
- List changes to reflect starting address of new free block
  - Example: 7560 was smallest beginning address
- Sizes of the three free partitions must be combined
  - Example: (20 + 20 + 205)
- Combined entry (last of the three) given status of "null"
  - Example: 7600

Case 2: Joining Three Free Blocks (cont'd.)

Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	245	Free
*		(null entry)
10250	4050	Free
15125	230	Free
24500	1000	Free

(table 2.7)
Case 2. The free list after a job has released memory.

Case 3: Deallocating an Isolated Block

- Deallocated memory space
  - Isolated from other free areas
- System determines released memory block status
  - Not adjacent to any free blocks of memory
  - Between two other busy areas
- System searches table for a null entry
  - Occurs when memory block between two other busy memory blocks is returned to the free list

Case 3: Deallocating an Isolated Block (cont'd.)

Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	245	Free
		(null entry)
10250	4050	Free
15125	230	Free
24500	1000	Free

(table 2.8)
Case 3. Original free list before deallocation. The soon-to-be-free memory block (at location 8805) is not adjacent to any blocks that are already free.

Case 3: Deallocating an Isolated Block (cont'd.)

Beginning Address	Memory Block Size	Status
7805	1000	Busy
*8805	445	Busy
9250	1000	Busy

(table 2.9)
Case 3. Busy memory list before deallocation. The job to be deallocated is of size 445 and begins at location 8805. The asterisk indicates the soon-to-be-free memory block.

Case 3: Deallocating an Isolated Block (cont'd.)

Beginning Address	Memory Block Size	Status
7805	1000	Busy
*		(null entry)
9250	1000	Busy

(table 2.10)
Case 3. This is the busy list after the job has released its memory. The asterisk indicates the new null entry in the busy list.

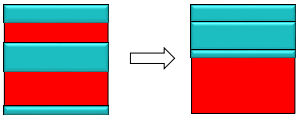
Case 3: Deallocating an Isolated Block (cont'd.)

Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	245	Free
*8805	445	Free
10250	4050	Free
15125	230	Free
24500	1000	Free

(table 2.11)
Case 3. This is the free list after the job has released its memory. The asterisk indicates the new free block entry replacing the null entry.

Relocatable Dynamic Partitions

- Memory Manager relocates programs
  - Gathers together all empty blocks
- Compact the empty blocks
  - Make one block of memory large enough to accommodate some or all of the jobs waiting to get in (Garbage collection or defragmentation)



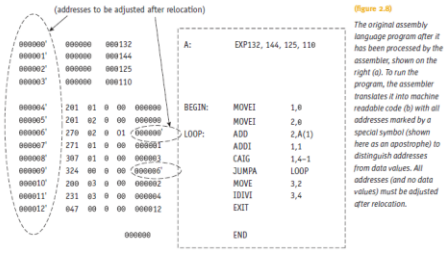
Relocatable Dynamic Partitions (cont'd.)

- **Compaction:** reclaiming fragmented sections of memory space
  - Every program in memory must be relocated
    - Programs become contiguous
  - Operating system must distinguish between addresses and data values
    - Every address adjusted to account for the program's new location in memory
    - Data values left alone

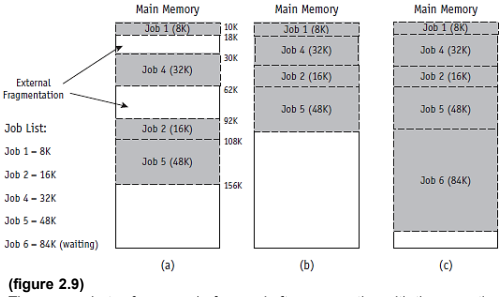

```
A       EXP 132, 144, 125, 110      ;the data values
BEGIN:  MOVEI           1,0         ;initialize register 1
        MOVEI           2,0         ;initialize register 2
LOOP:   ADD             2,A(1)      ;add (A + reg 1) to reg 2
        ADDI            1,1         ;add 1 to reg 1
        CAIG            1,4-1       ;is register 1 > 4-1?
        JUMPA           LOOP        ;if not, go to Loop
        MOVE            3,2         ;if so, move reg 2 to reg 3
        IDIVI           3,4         ;divide reg 3 by 4,
                                    ;remainder to register 4
        EXIT                        ;end
        END
```

(figure 2.7)
An assembly language program that performs a simple incremental operation. This is what the programmer submits to the assembler. The commands are shown on the left and the comments explaining each command are shown on the right after the semicolons.

Relocatable Dynamic Partitions (cont'd.)



(figure 2.8)
The original assembly language program after it has been processed by the assembler, shown on the right (a). To run the program, the assembler translates it into machine readable code (b) with all addresses marked by a special symbol (shown here as an apostrophe) to distinguish addresses from data values. All addresses (and no data values) must be adjusted after relocation.

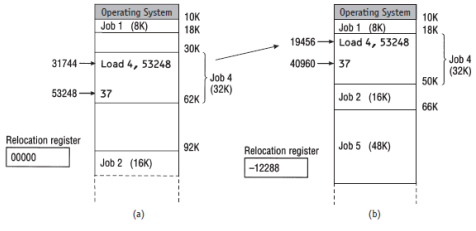


(figure 2.9)
Three snapshots of memory before and after compaction with the operating system occupying the first 10K of memory. When Job 6 arrives requiring 84K, the initial memory layout in (a) shows external fragmentation totaling 96K of space. Immediately after compaction (b), external fragmentation has been eliminated, making room for Job 6 which, after loading, is shown in (c).

Relocatable Dynamic Partitions (cont'd.)

- Special-purpose registers used for relocation:
  - Bounds register
    - Stores highest location accessible by each program to avoid out of the bound
  - Relocation Register
    - Contains the value that must be added to each address

Relocatable Dynamic Partitions (cont'd.)



(figure 2.10)
Contents of relocation register and close-up of Job 4 memory area (a) before relocation and (b) after relocation and compaction.

Relocatable Dynamic Partitions (cont'd.)

- Compaction issues:
  - What goes on behind the scenes when relocation and compaction take place?
  - What keeps track of how far each job has moved from its original storage area?
  - What lists have to be updated?

Relocatable Dynamic Partitions (cont'd.)

- **What lists have to be updated?**
  - Free list
    - Must show the partition for the new block of free memory
  - Busy list
    - Must show the new locations for all of the jobs already in process that were relocated
  - Each job will have a new address
    - Exception: those already at the lowest memory locations

Relocatable Dynamic Partitions (cont'd.)

- Special-purpose registers used for relocation:
  - **Bounds register**
    - Stores highest location accessible by each program
  - **Relocation register**
    - Contains the value that must be added to each address referenced in the program
    - Must be able to access the correct memory addresses after relocation
    - If the program is not relocated, "zero" value stored in the program's relocation register

Relocatable Dynamic Partitions (cont'd.)

- Compacting and relocating optimizes use of memory
  - Improves throughput
- Options for timing of compaction:
  - When a certain percentage of memory is busy
  - When there are jobs waiting to get in
  - After a prescribed amount of time has elapsed
- Compaction entails more overhead
- **Goal:** optimize processing time and memory use while keeping overhead as low as possible

Summary

- Four memory management techniques
  - Single-user systems, fixed partitions, dynamic partitions, and relocatable dynamic partitions
- Common requirements of four memory management techniques
  - Entire program loaded into memory
  - Contiguous storage
  - Memory residency until job completed
- Each places severe restrictions on job size
- Sufficient for first three generations of computers

Summary (cont'd.)

- New modern memory management trends in late 1960s and early 1970s
  - Discussed in next chapter
  - Common characteristics of memory schemes
    - Programs are not stored in contiguous memory
    - Not all segments reside in memory during job execution