Chapter 2
Memory Management: Early Systems

Understanding Operating Systems, Fourth Edition

Objectives

You will be able to describe:

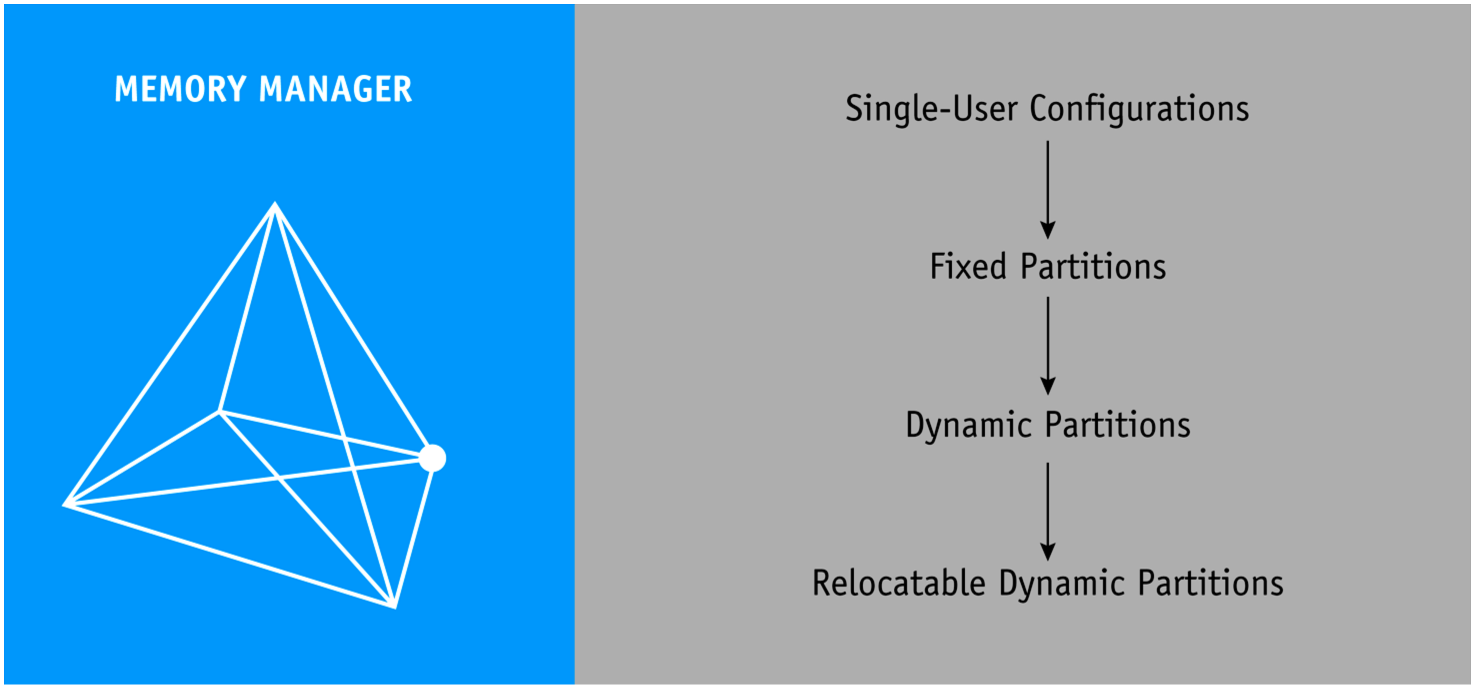
- The basic functionality of the three memory allocation schemes presented in this chapter: fixed partitions, dynamic partitions, relocatable dynamic partitions
- Best-fit memory allocation as well as first-fit memory allocation schemes
- How a memory list keeps track of available memory
- The importance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

Objectives (continued)

Students should be able to describe:

- The importance of the bounds register in memory allocation schemes
- The role of compaction and how it improves memory allocation efficiency

Memory Management: Early Systems



"Memory is the primary and fundamental power, without which there could be no other intellectual operation." —Samuel Johnson (1709–1784)

Memory Management: Early Systems

- **Types of memory allocation schemes:**
  - Single-user systems
  - Fixed partitions
  - Dynamic partitions
  - Relocatable dynamic partitions

Single-User Contiguous Scheme

- **Single-User Contiguous Scheme:** Program is loaded in its entirety into memory and allocated as much contiguous space in memory as it needs
  - Jobs processed sequentially in single-user systems
  - Requires minimal work by the Memory Manager
    - Register to store the base address
    - Accumulator to keep track of the program size

Single-User Contiguous Scheme (continued)

- **Disadvantages of Single-User Contiguous Scheme:**
  - Doesn’t support multiprogramming
  - Not cost effective

Fixed Partitions

- **Fixed Partitions:** Main memory is partitioned; one partition/job
  - Allows multiprogramming
  - Partition sizes remain static unless and until computer system id shut down, reconfigured, and restarted
  - Requires protection of the job’s memory space
  - Requires matching job size with partition size

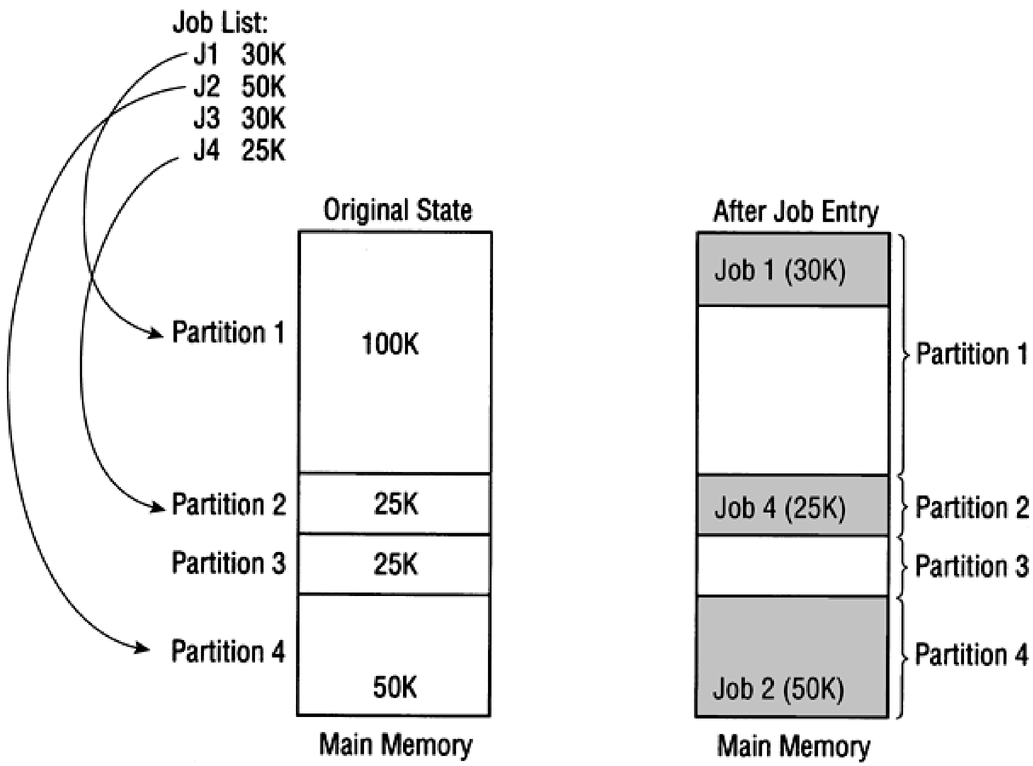
Fixed Partitions (continued)

To allocate memory spaces to jobs, the operating system's Memory Manager must keep a table as shown below:

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

Fixed Partitions (continued)



NOTE: Job 3 must wait even though 70K of free space is available in Partition 1 where Job 1 occupies only 30K of the 100K available

Figure 2.1: Main memory use during fixed partition allocation of Table 2.1

Fixed Partitions (continued)

- **Disadvantages:**
  - Requires entire program to be stored contiguously
  - Jobs are allocated space on the basis of first available partition of required size
  - Works well only if all of the jobs are of the same size or if the sizes are known ahead of time
  - Arbitrary partition sizes lead to undesired results
    - Too small a partition size results in large jobs having longer turnaround time
    - Too large a partition size results in memory waste or internal fragmentation

Dynamic Partitions

- **Dynamic Partitions:** Jobs are given only as much memory as they request when they are loaded
  – Available memory is kept in contiguous blocks
  – Memory waste is comparatively small
- **Disadvantages:**
  – Fully utilizes memory only when the first jobs are loaded
  – Subsequent allocation leads to memory waste or external fragmentation

Dynamic Partitions (continued)

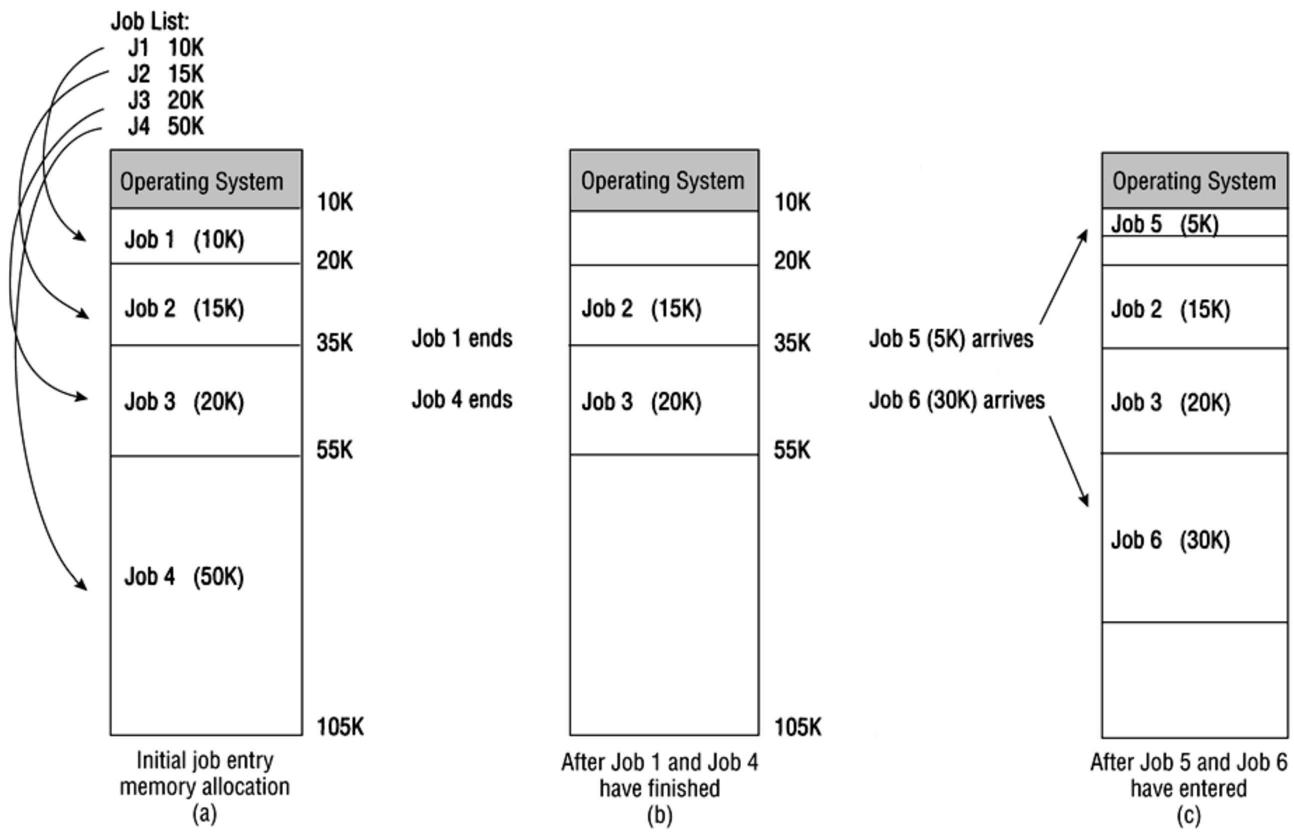


Figure 2.2: Main memory use during dynamic partition allocation

Dynamic Partitions (continued)

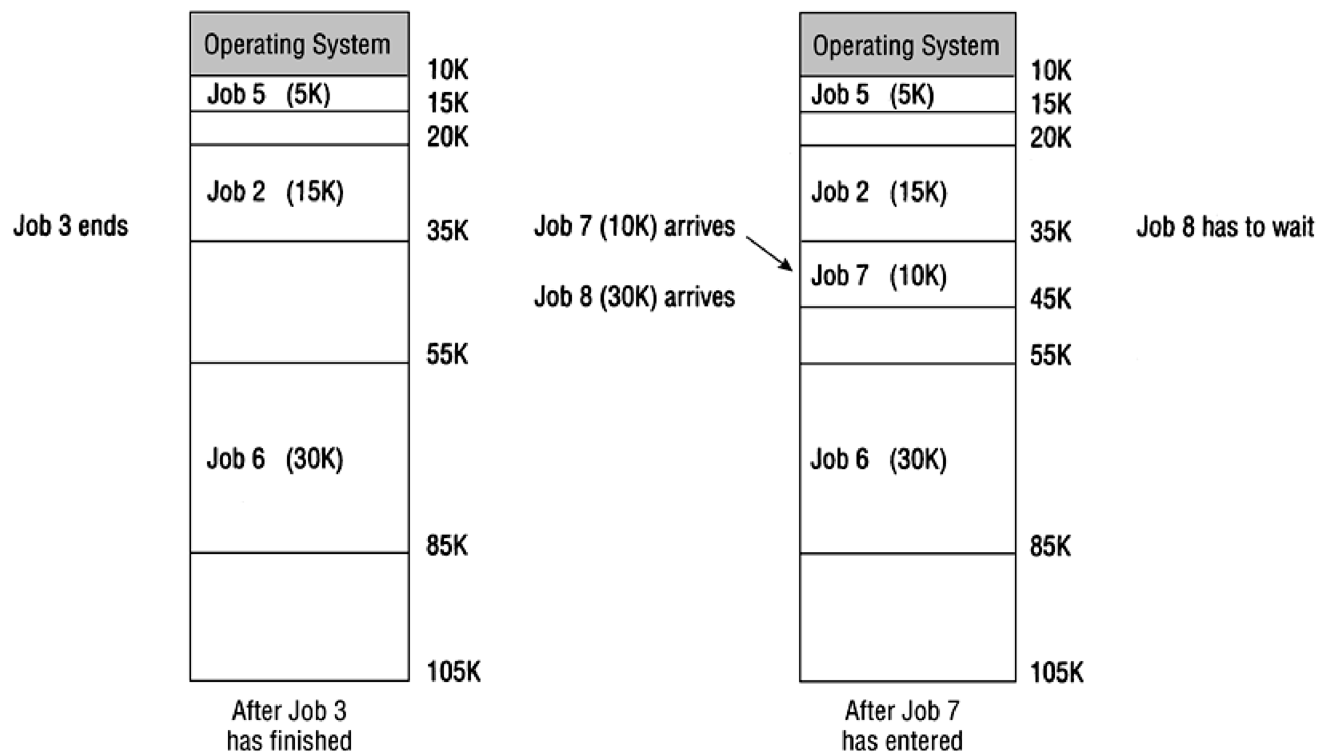


Figure 2.2 (continued): Main memory use during dynamic partition allocation

Best-Fit Versus First-Fit Allocation

- Free partitions are allocated on the following basis:
  - **First-fit memory allocation:** First partition fitting the requirements
    - Leads to fast allocation of memory space
  - **Best-fit memory allocation:** Smallest partition fitting the requirements
    - Results in least wasted space
    - Internal fragmentation reduced but not eliminated

Best-Fit Versus First-Fit Allocation (continued)

- **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 (continued)

Job List:

Job number	Memory requested
J1	10K
J2	20K
J3	30K*
J4	10K

Memory List:

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
10240	30K	J1	10K	Busy	20K
40960	15K	J4	10K	Busy	5K
56320	50K	J2	20K	Busy	30K
107520	20K			Free	
Total Available:	115K	Total Used:	40K		

Figure 2.3: An example of a first-fit free scheme

Best-Fit Versus First-Fit Allocation (continued)

Job List:

Job number	Memory requested
J1	10K
J2	20K
J3	30K
J4	10K

Memory List:

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
40960	15K	J1	10K	Busy	5K
107520	20K	J2	20K	Busy	None
10240	30K	J3	30K	Busy	None
56320	50K	J4	10K	Busy	40K
Total Available:	115K	Total Used:	70K		

Figure 2.4: An example of a best-fit free scheme

Best-Fit Versus First-Fit Allocation (continued)

- **Algorithm for First-Fit:**
  - Assumes Memory Manager keeps two lists, one for free memory and one for busy memory blocks
  - Loop compares the size of each job to the size of each memory block until a block is found that's large enough to fit the job
  - Job is stored into that block of memory
  - Memory Manager moves out of the loop to fetch the next job from the entry queue

Best-Fit Versus First-Fit Allocation (continued)

- **Algorithm for First-Fit (continued):**
  - If the entire list is searched in vain, then the job is placed into a waiting queue
  - The Memory Manager then fetches the next job and repeats the process

Best-Fit Versus First-Fit Allocation (continued)

Before Request		After Request	
Beginning Address	Memory Block Size	Beginning Address	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: Status of each memory block before and after a request is made for a block of 200 spaces using the first-fit algorithm

Best-Fit Versus First-Fit Allocation (continued)

- **Algorithm for Best-Fit:**
  - Goal: find the smallest memory block into which the job will fit
  - Entire table must be searched before allocation

Best-Fit Versus First-Fit Allocation (continued)

Before Request		After Request	
Beginning Address	Memory Block Size	Beginning Address	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: Status of each memory block before and after a request is made for a memory block of 200 spaces using the best-fit algorithm

Best-Fit Versus First-Fit Allocation (continued)

- **Hypothetical allocation schemes:**
  - **Next-fit:** Starts searching from last allocated block, for the next available block when a new job arrives
  - **Worst-fit:** Allocates the largest free available block to the new job
    - Opposite of best-fit
    - Good way to explore the theory of memory allocation; might not be the best choice for an actual system

Deallocation

- **Deallocation:** Freeing an allocated memory space
  - **For fixed-partition system**:
    - Straightforward process
    - When job completes, Memory Manager resets the status of the job's memory block to "free"
    - Any code—for example, binary values with 0 indicating free and 1 indicating busy—may be used

Deallocation (continued)

- **For dynamic-partition system**:
  - Algorithm tries to combine free areas of memory whenever possible
  - Three cases:
    - **Case 1:** When the block to be deallocated is adjacent to another free block
    - **Case 2:** When the block to be deallocated is between two free blocks
    - **Case 3:** When the block to be deallocated is isolated from other free blocks

Deallocation: Dynamic Partition System

- **Case 1: Joining Two Free Blocks**
  - Change list must reflect starting address of the new free block
    - In the example, 7600—which was the address of the first instruction of the job that just released this block
  - Memory block size for the new free space must be changed to show its new size—that is, the combined total of the two free partitions
    - In the example, (200 + 5)

Case 1: Joining Two Free Blocks

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.

Table 2.4: Original free list before deallocation for Case 1

Case 1: Joining Two Free Blocks (continued)

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: Free list after deallocation for Case 1

Deallocation: Dynamic Partition System (continued)

- **Case 2: Joining Three Free Blocks.** Deallocated memory space is between two free memory blocks
  - Change list to reflect the starting address of the new free block
    - In the example, 7560— which was the smallest beginning address
  - Sizes of the three free partitions must be combined
    - In the example, (20 + 20 + 205)
  - Combined entry is given the status of null entry
    - In the example, 7600

Case 2: Joining Three Free Blocks

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: Original free list before deallocation for Case 2

Case 2: Joining Three Free Blocks (continued)

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: Free list after job has released memory

Deallocation: Dynamic Partition System (continued)

- **Case 3: Deallocating an Isolated Block.** Space to be deallocated is isolated from other free areas
  - System learns that the memory block to be released is not adjacent to any free blocks of memory, it is between two other busy areas
  - Must search the table for a null entry
  - Null entry in the busy list occurs when a memory block between two other busy memory blocks is returned to the free list

Case 3: Deallocating an Isolated Block

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: Original free list before deallocation for Case 3

Case 3: Deallocating an Isolated Block (continued)

The job to be deallocated is of size 445 and begins at location 8805. The asterisk indicates the soon-to-be-free memory block.

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

Table 2.9: Memory list before deallocation

Case 3: Deallocating an Isolated Block (continued)

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

Table 2.10: 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 (continued)

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: Free list after the job has released its memory. The asterisk indicates the new free block entry replacing the null entry

Relocatable Dynamic Partitions

- **Relocatable Dynamic Partitions:**
  - Memory Manager relocates programs to gather together all of the empty blocks
  - Compact the empty blocks to make one block of memory large enough to accommodate some or all of the jobs waiting to get in

Relocatable Dynamic Partitions (continued)

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

Relocatable Dynamic Partitions (continued)

```
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 reg 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 2.5: An assembly language program that performs a simple incremental operation

Relocatable Dynamic Partitions (continued)

```
000000' 000000 000132        A:       EXP132,144,125,110
000001' 000000 000144
000002' 000000 000125
000003' 000000 000110

000004' 201 01 0 00 000000   BEGIN:   MOVEI     1,0
000005' 201 02 0 00 000000            MOVEI     2,0
000006' 270 02 0 01 000000'  LOOP:    ADD       2,A(1)
000007' 271 01 0 00 000001            ADDI      1,1
000008' 307 01 0 00 000003            CAIG      1,4-1
000009' 324 00 0 00 000006'           JUMPA     LOOP
000010' 200 03 0 00 000002            MOVE      3,2
000011' 231 03 0 00 000004            IDIVI     3,4
000012' 047 00 0 00 000012            EXIT

                  000000              END
```

Figure 2.6: The original assembly language program after it has been processed by the assembler

Relocatable Dynamic Partitions (continued)

- 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 (continued)

- **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 except for those that were already at the lowest memory locations

Relocatable Dynamic Partitions (continued)

- Special-purpose registers are 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 so it will be able to access the correct memory addresses after relocation
    - If the program isn't relocated, the value stored in the program's relocation register is zero

Relocatable Dynamic Partitions (continued)

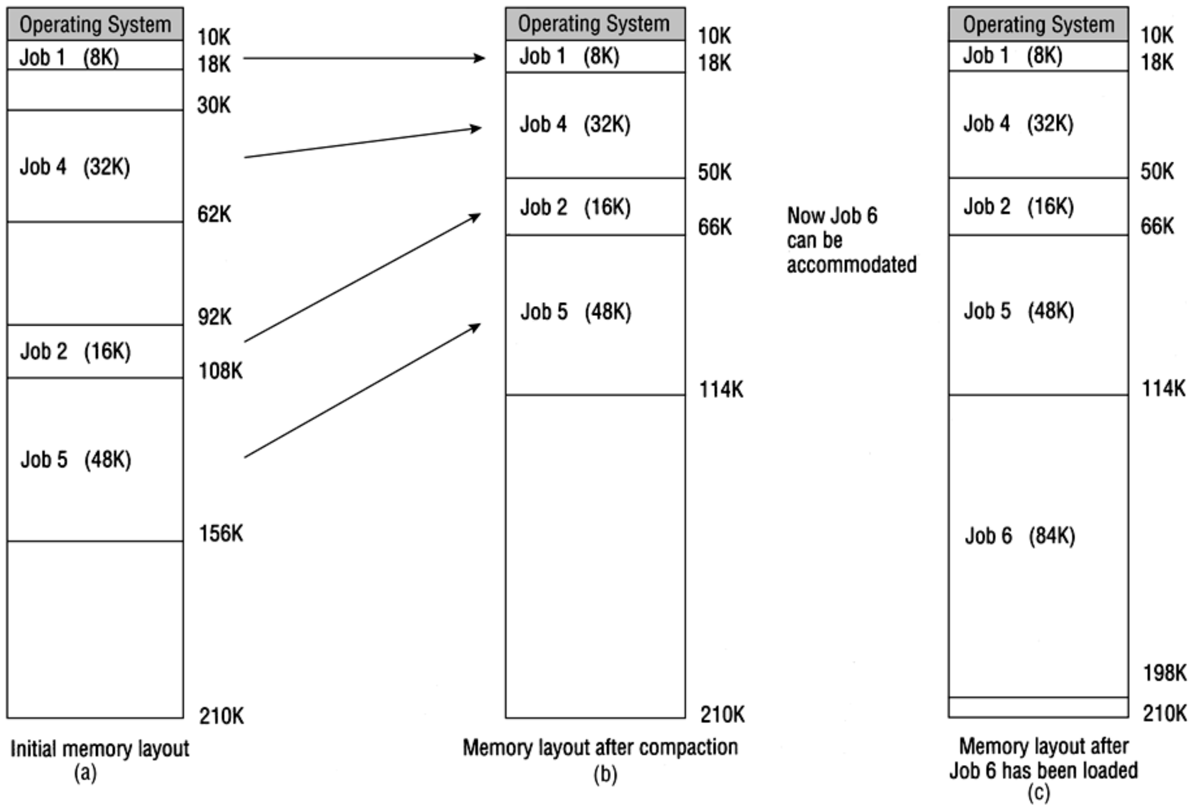


Figure 2.7: Three snapshots of memory before and after compaction

Relocatable Dynamic Partitions (continued)

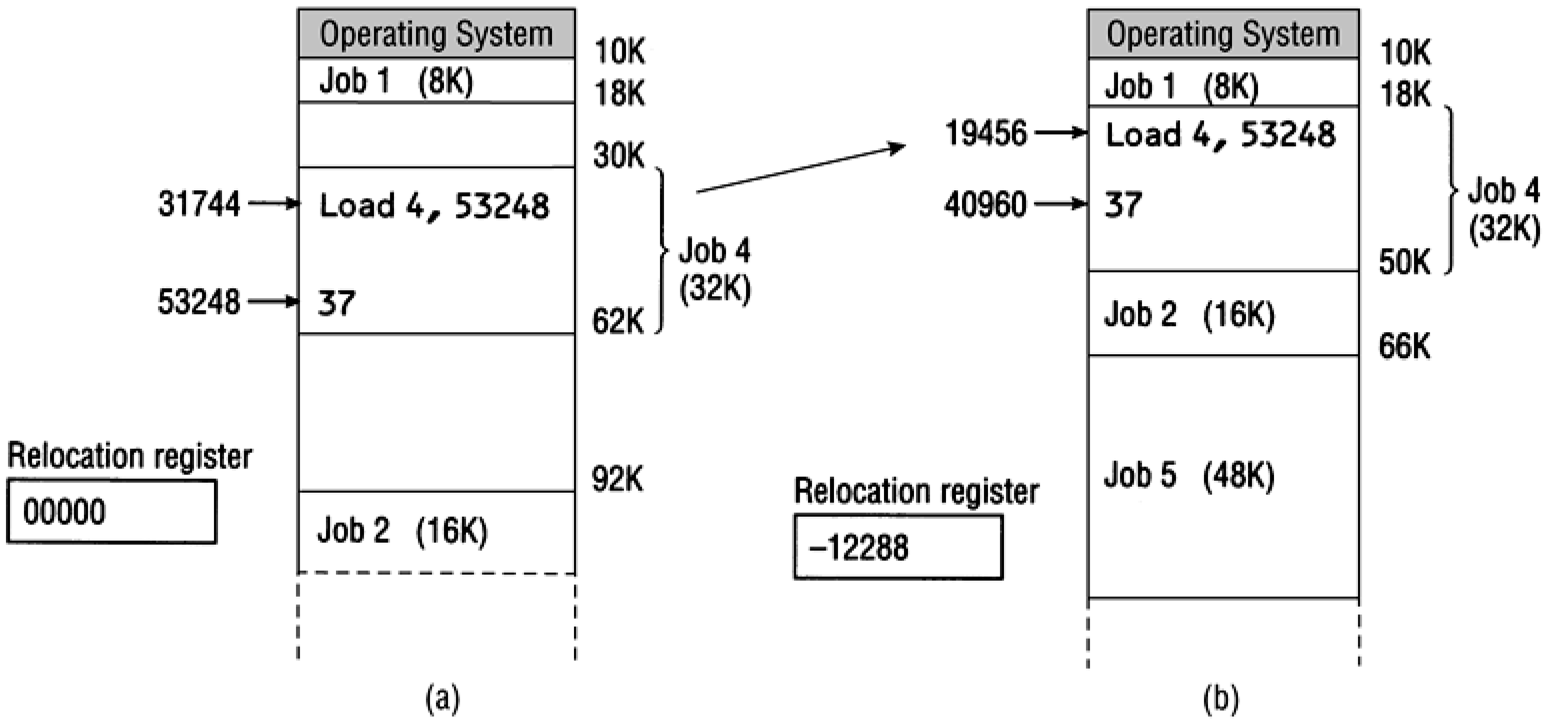


Figure 2.8: Contents of relocation register and close-up of Job 4 memory area (a) before relocation and (b) after relocation and compaction

Relocatable Dynamic Partitions (continued)

- Compacting and relocating optimizes the use of memory and thus improves throughput
- Options for when and how often it should be done:
  - When a certain percentage of memory is busy
  - When there are jobs waiting to get in
  - After a prescribed amount of time has elapsed

  Goal: Optimize processing time and memory use while keeping overhead as low as possible

Summary

- Four memory management techniques were used in early systems: single-user systems, fixed partitions, dynamic partitions, and relocatable dynamic partitions
- Memory waste in dynamic partitions is comparatively small as compared to fixed partitions
- First-fit is faster in making allocation but leads to memory waste
- Best-fit makes the best use of memory space but slower in making allocation

Summary (continued)

- Compacting and relocating optimizes the use of memory and thus improves throughput
- All techniques require that the entire program must:
  - Be loaded into memory
  - Be stored contiguously
  - Remain in memory until the job is completed
- Each technique puts severe restrictions on the size of the jobs: can only be as large as the largest partitions in memory