UNIVERSITI TENAGA NASIONAL

Memory Management

CGMB143 COMPUTER SYSTEM

INTRODUCTION TO MEMORY MANAGEMENT (MMR)

Memory Management

- In uniprogramming system, memory is divided into two parts:
  - for the OS
  - for the program that currently being executed – user part
- In multiprogramming system, the user part of the memory must be further subdivided to accommodate multiple processes.
- The task of subdivision is carried out dynamically by the OS → **memory management**.

Memory Management: cont

- Memory needs to be allocated efficiently to pack as many processes into memory as possible.
- This is because effective memory management is important in multiprogramming system

Memory Management Requirements

- Five requirements of memory management:
  1. Relocation
  2. Protection
  3 Sharing
  4. Logical Organization
  5. Physical Organization

RELOCATION

MMR: 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 memory at a different location (relocate process at different area)
- The location for the process is unknown and the process must be allowed to moved in and out due to swapping.

MMR: Relocation

- Memory references must be translated to the actual physical memory address

PROTECTION

MMR: Protection

- Processes should not be able to reference memory locations in another process without permission.
- Impossible to check absolute addresses in programs since the program could be relocated.
- All memory references must be checked at run time to ensure that it refer to the only memory space allocated to the process

SHARING

MMR: Sharing

- Allow several processes to access the same portion of memory.
- So, any protection mechanism must have the flexibility, example:
  - If a number of processes are executing the same program, it is advantageous to allow each process access to the same copy of the program rather than have their own separate copy → e.g. ECHO procedure

MMR: Sharing

- Memory management system must therefore allow controlled access to shared areas of memory without compromising essential protection.

ORGANIZATION

MMR: 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

MMR: 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.

Memory Management

- Principal operation of memory management is to bring programs into memory for execution by the processor.
- In most modern multiprogramming system it invokes virtual memory (VM) that use both segmentation and paging techniques.
- First look at simpler technique that do not use Virtual Memory
  - Partitioning
  - Simple Paging
  - Simple Segmentation

PARTITIONING

Memory Partitioning

- Divide the memory to small size partition.
- Two types of partitioning:
    1. Fixed Partitioning
    2. Dynamic Partitioning

Fixed Partitioning

- Most schemes of memory management, assume that
  - the OS occupies some fixed partition of memory
  - the rest of memory is available for use by multiple processes. →USER AREA
- The simplest scheme for managing this available memory is to partition it into regions with fixed boundaries.
- Figure 7.2 shows example of two alternatives for fixed partitioning:
  1. Equal-size partition
  2. Unequal-size partition

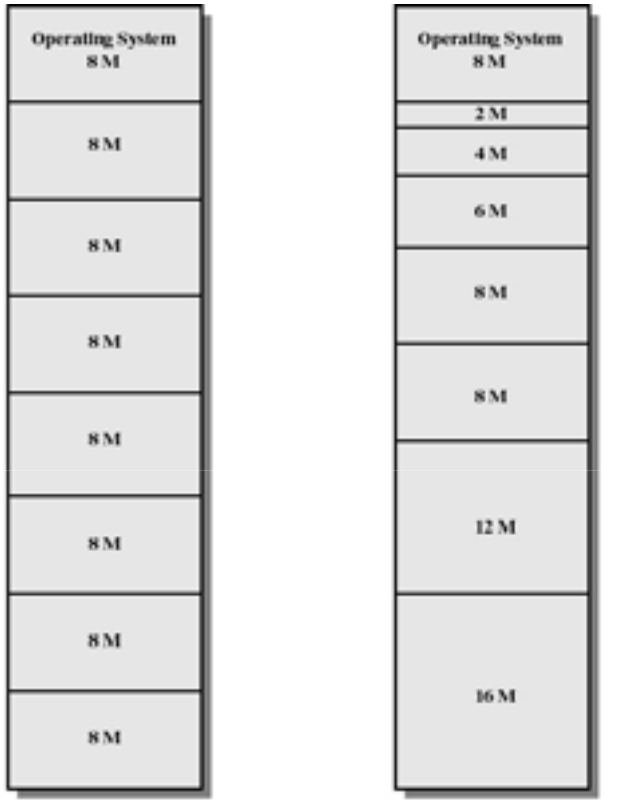


Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

Fixed Partitioning

- Equal-size partition (Figure 7.2a)
  - Divide the memory into same size partition
  - 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

Fixed Partitioning

- Two difficulties:
  - A program may not fit in a partition.

    – The programmer must design the program with "overlays".
  - Memory use is inefficient.

    – Any program, no matter how small, occupies an entire partition. This is called internal fragmentation.

    » E.g. →Program length 2MB, it occupies an 8MB partition. So 6MB is internal fragmentation.

Fixed Partitioning

- Unequal-size partition (Figure 7.2b)
  - Divide the memory into different partition size.
  - A program as large as 16MB can be accommodate without need to overlays it.
  - Partition smaller than 8MB allow smaller program to be accommodate with less internal fragmentation.

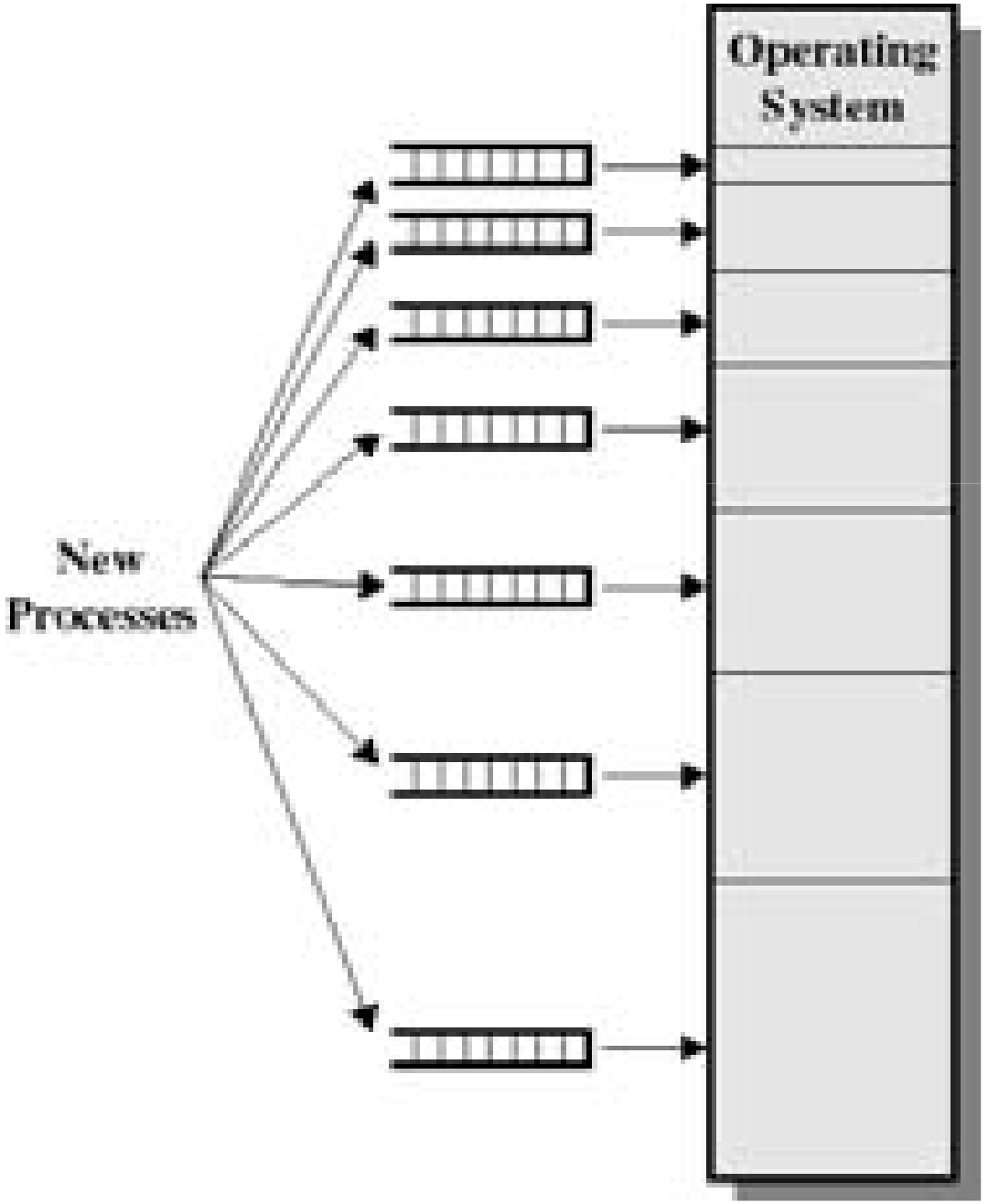
Fixed Partitioning - Placement Algorithm

Where to put / load process in memory.

- Equal-size partitions
  - If there is an available partition, a process can be loaded into that partition
    - because all partitions are of equal size it does not matter which partition is used
  - If all partitions are occupied by blocked processes, choose one process to swap out to make room for the new process

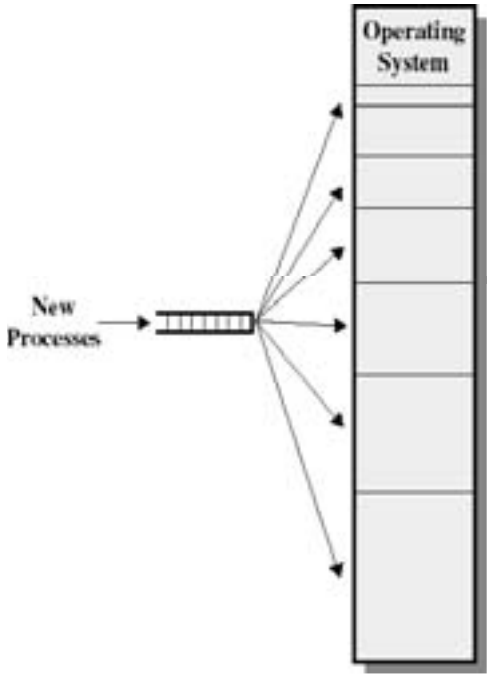
Fixed Partitioning - Placement Algorithm

- Unequal-size partitions:
  - Two ways:
    1. Use of multiple queues
       - Assign each process to the smallest partition within which it will fit
       - A queue for each partition size
       - Tries to minimize internal fragmentation
       - Problem: some queues will be empty if no processes within a size range is present


Operating System
New Processes

Fixed Partitioning - Placement Algorithm

- 2. Use of a single queue
  - When its time to load a process into memory the smallest available partition that will hold the process is selected
  - If all partitions is occupied:
    - Preference to swapping out the smallest partition that will hold the incoming process.
    - Also have to consider other factors such as priority and blocked vs. ready process.
- Increases the level of multiprogramming at the expense of internal fragmentation.


Operating System
New Processes

Fixed Partitioning

- The used of unequal-size partition provides a degree of flexibility to fixed partitioning.
- Fixed partitioning schemes are relatively simple and require minimal OS software and processing overhead.
- But it still have disadvantages:

  1. The number of partitions specified at system generation time limits the number of active (not suspended) processes in the system.

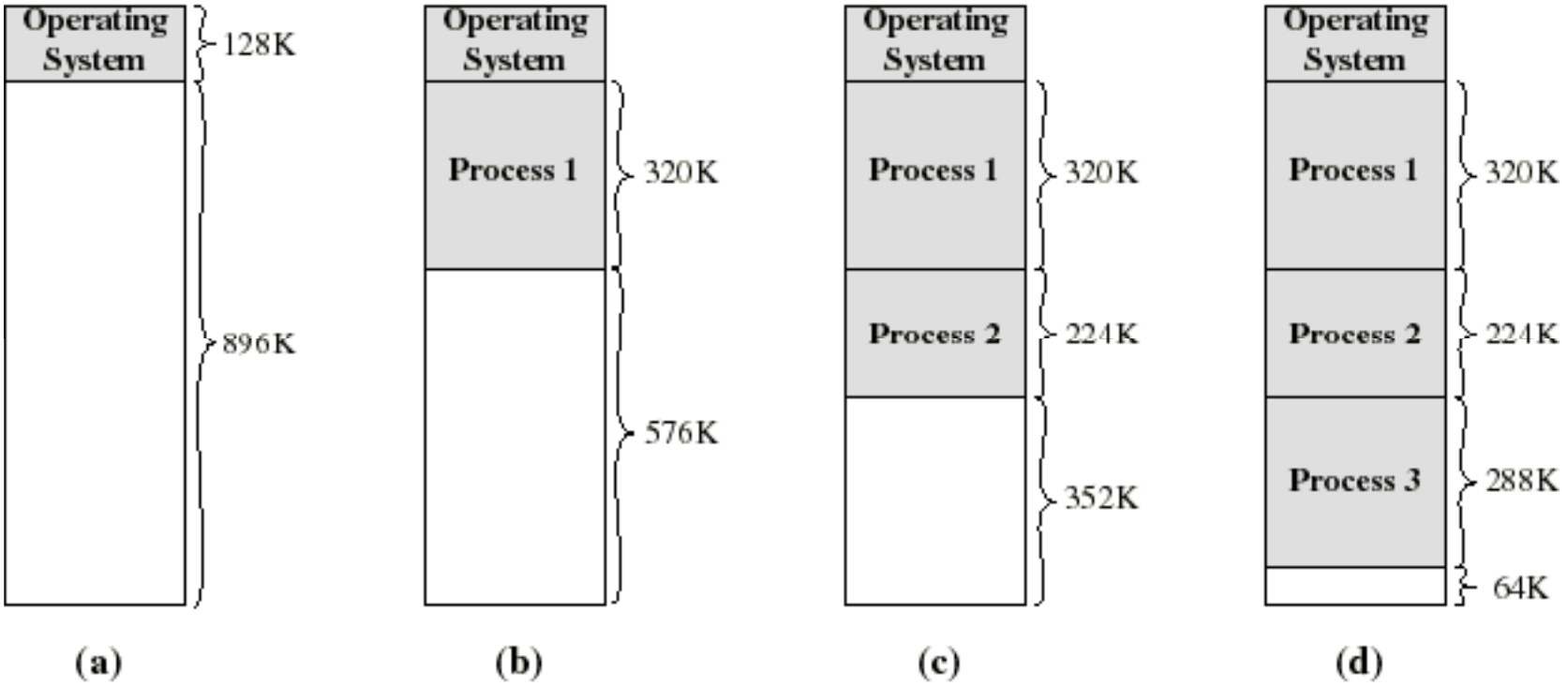
  2. Because partition sizes are preset at system generation time, small jobs will not utilize partition space efficiently.

Fixed Partitioning

- The used of fixed partitioning is almost unknown today.
- Example of OS that use this technique was an early IBM mainframe OS, OS/MFT (Multiprogramming with a Fixed Number of Tasks).

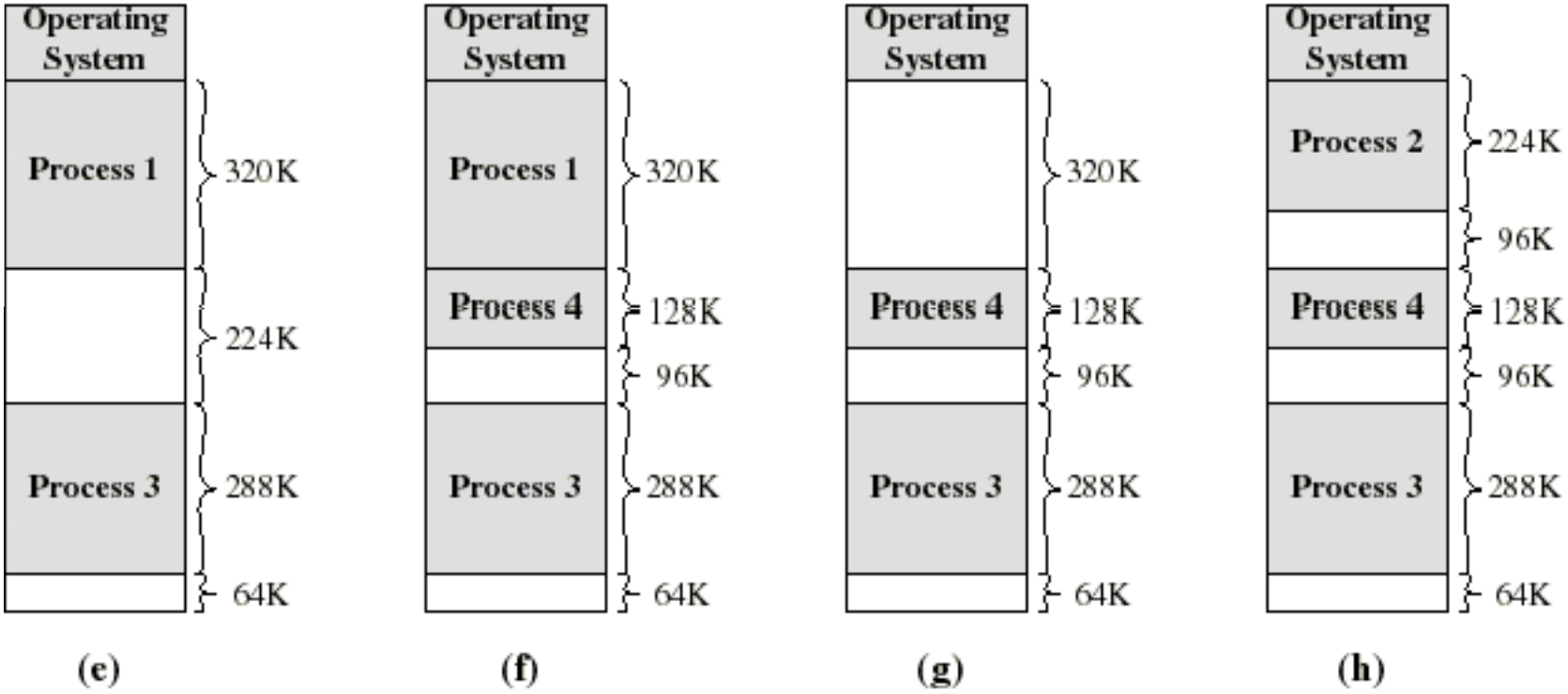
Dynamic Partitioning

- Partitions are of variable length and number
- Each process is allocated exactly as much memory as it requires.
- Used in IBM's OS/MVT (Multiprogramming with a Variable number of Tasks)
- An example using 1 MB (1024KB) of memory is shown in Figure 7.4.


Operating System
128K
896K
(a)
Operating System
Process 1
320K
576K
(b)
Operating System
Process 1
320K
Process 2
224K
352K
(c)
Operating System
Process 1
320K
Process 2
224K
Process 3
288K
64K
(d)

Dynamic Partitioning

- Initially memory is empty, except for the OS(refer figure a)
- The first three process are loaded in, starting where the OS ends and occupying just enough space for each process (refer figure b,c,d).
- This leaves a “hole” at the end of memory that is too small for a fourth process.


Operating System
Process 1
320K
224K
Process 3
288K
64K
(e)
Operating System
Process 1
320K
Process 4
128K
96K
Process 3
288K
64K
(f)
Operating System
320K
Process 4
128K
96K
Process 3
288K
64K
(g)
Operating System
Process 2
224K
96K
Process 4
128K
96K
Process 3
288K
64K
(h)

Dynamic Partitioning

- At some point none of the processes in memory is ready. OS swaps process 2 (figure e), which leaves sufficient room to load a new process, process 4 (figure f). Because process 4 is smaller than process 2, another small hole is created.

- Later, a point is reached at which none of the processes in memory is ready, but process 2, in the Ready-Suspend state is available.

- So OS swaps process 1 out (figure g) and swaps process 2 back in (figure h).

Dynamic Partitioning

- Eventually holes are formed in the memory.
- Memory will become more and more fragmented, and memory utilization declines.
- This hole is called external fragmentation.
  - Referring to fact that the memory is external to all partitions becomes increasingly fragmented.
- Technique to overcome external fragmentation is called **compaction**.
  - Time to time OS shifts the processes so that they are contiguous and so that all of free memory is together in one block.

Dynamic Partitioning

- Example in figure 7.4h, compaction will result in a block of free memory of length 256K, and this maybe sufficient to load an additional process.
- Difficulties of compaction is time consuming procedure and wasteful of processor time.

Dynamic Partitioning - Placement Algorithm

→ Where to put / load process in memory.

- OS must decide which free block to allocate to a process.
- 3 placement algorithms that might be considered:
  1. Best Fit
  2. First Fit
  3. Next Fit

Dynamic Partitioning - Placement Algorithm

1. Best Fit
   - Chooses the block that is closest in size to the request.
   - Chooses smallest hole
2. First fit:
   - Begins to scan memory from the beginning and chooses the first available block that is large enough.
   - Chooses first hole from beginning

Dynamic Partitioning - Placement Algorithm

3. Next-fit:
   - Begins to scan memory from the last placement and chooses the next available block that is large enough.
   - Chooses first hole from last placement

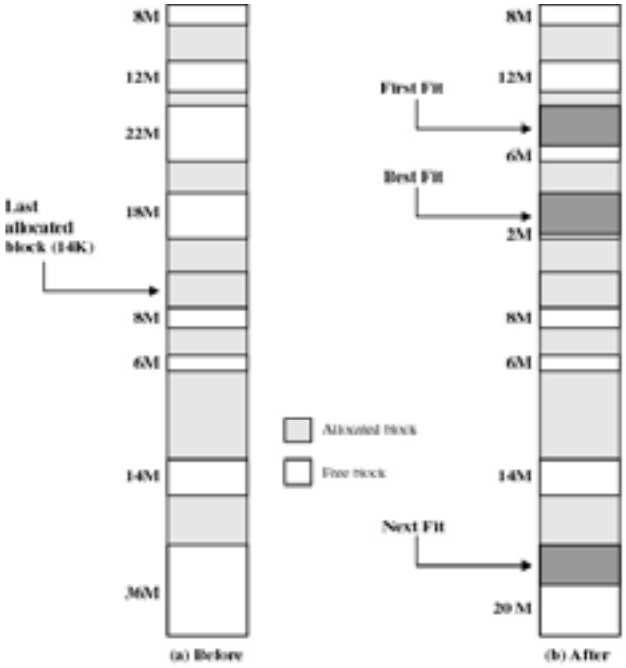


Figure 7.5 Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Dynamic Partitioning - Placement Algorithm

- Which of these algorithms is best will depend on the exact sequence of process swapping that occurs and the size of those processes.
- Remarks for each of the algorithms:
  1. First Fit
     - The simplest and usually the best and fastest.
  2. Next-Fit
     - Tends to produce slightly worse result than first fit.
     - More frequently lead to an allocation from a free block at the end of memory.
     - Compaction maybe required more frequentlywith next fit

Dynamic Partitioning - Placement Algorithm

3. Best Fit

- Worse performer, because this algorithm will looks for the smallest block that will satisfy the requirement, it guarantees that the fragment left behind is small as possible.
- The memory is quickly littered by blocks too small to satisfy memory allocation requests.
- Compaction must be done more frequently than the other two algorithms.

Paging

Paging

- Partition memory into small equal-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 or page.
- Figure 7.9 illustrates the use of pages and frames.
- Some of the frames in memory are in use and some are free and list of

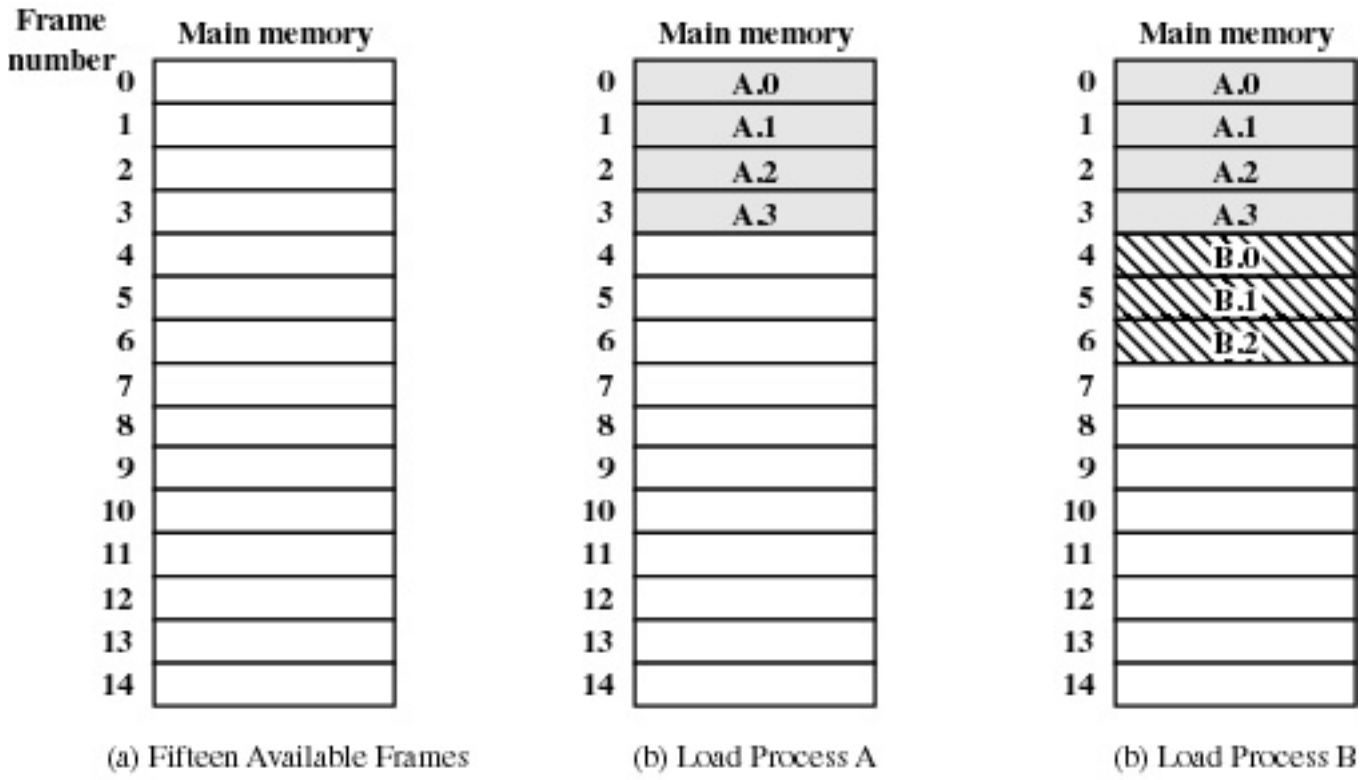


Figure 7.9 Assignment of Process Pages to Free Frames

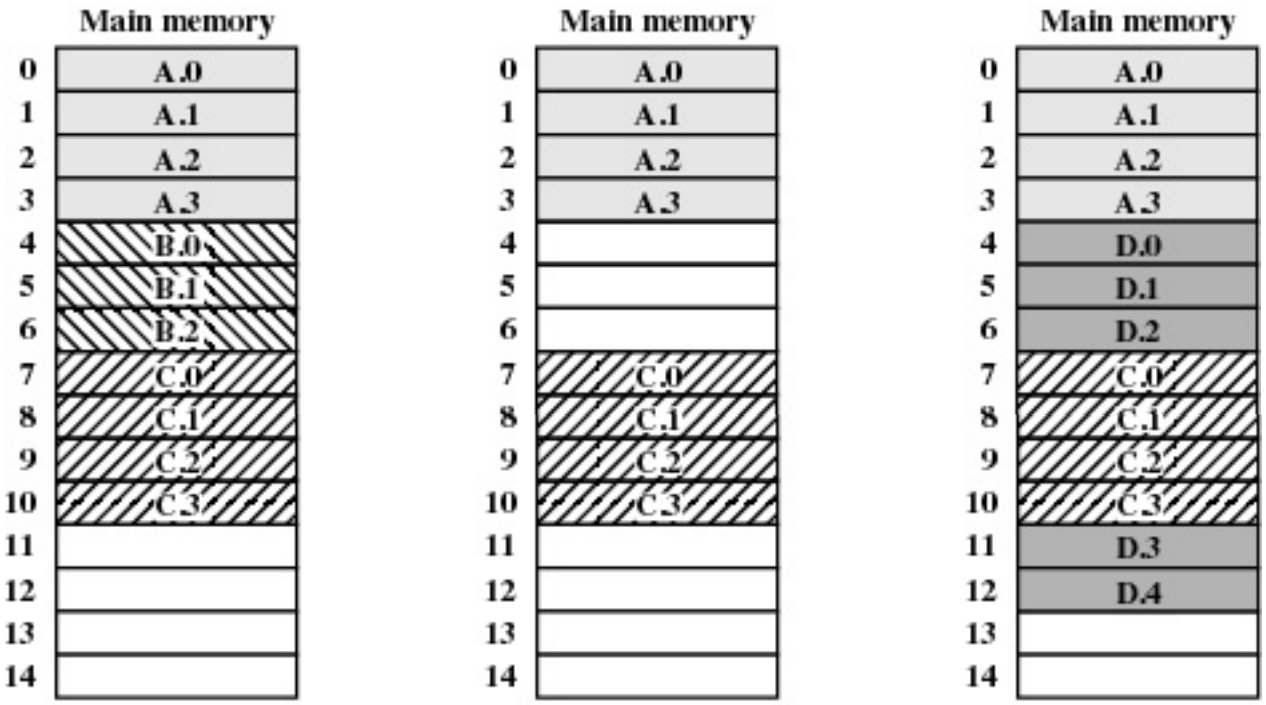


Figure 7.9 Assignment of Process Pages to Free Frames

Paging

- PA stored on disk, consists of four pages.
- When it comes to load PA, the OS finds four free frames and loads the four pages of PA into the four frames (Fig 7.9b).
- PB consisting of three pages and PC consisting of four pages are subsequently loaded. (Fig 7.9c,d).
- The PB is suspended and is swapped out of memory.

Paging

- Later all of the processes in memory are blocked, and the OS needs to bring in anew process, PD that contains five pages.
- From the figure (Fig7. 9e) we can see that there are no sufficient unused frames to hold the PD.
- Now the concept of logical address is used but with the absent of base register.
- OS maintains a page table for each process.
  - The page table shows the frames location of each page of the process.

Paging

- Within the program, each logical address consists of a page number and an offset within the page.
- Logical to physical address translation is done by processor and the processor must know how to access the page table of current process.
- Presented with a logical address (page #, offset), the processor uses the page table to produce a physical address (frame #, offset).
- From figure 7.9d, five pages of PD are loaded in frames 4, 5 ,6 ,11 and 12.

Paging

- Figure 7.10 shows the various page tables.
- Page table entry contains the number of the frame in memory, if any, that holds the corresponding page.
- OS also maintains a single free frame list of all frames in memory that are currently occupied and available pages.
- Simple paging similar to fixed partitioning but the different are:
  - In paging the partitions are rather small
  - Program may occupy more than one partitions and no need to be contiguous.

Page Tables for Example

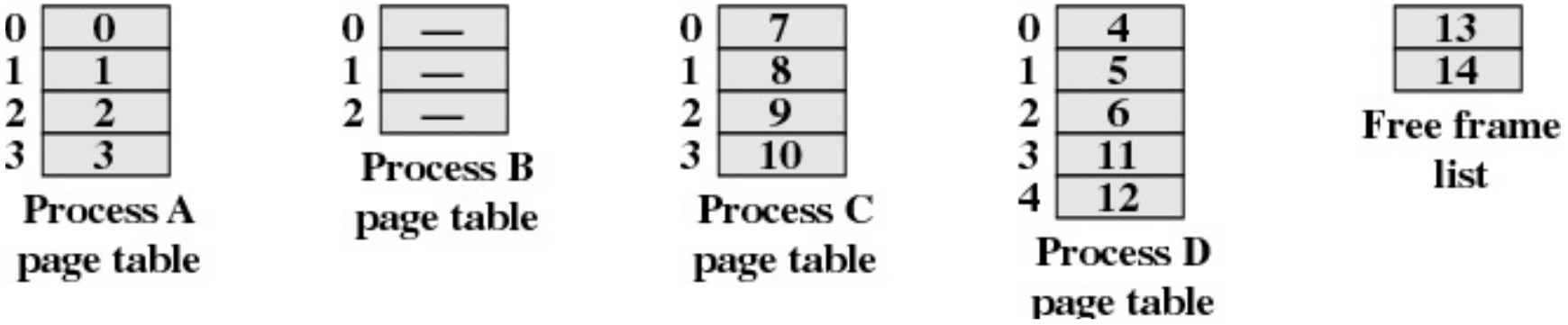


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Paging

- An example is shown in Figure 7.11
- In this example consider 16bit addresses are used, and the page size is 1K = 1024 bytes.
- The relative address 1502 in binary form is 0000010111011110
- Offset need 10 bits (m, rightmost) and page number 6 bits (n, leftmost). So the program can consist a maximum 26 = 64 pages of 1Kb each.
- As figure 7.11b shows address 1502 (0000010111011110) correspond to an offset of 478(0111011110) and page number 1 (000001).

Paging

- Consider an address of *n*+*m* bits, where the leftmost *n* bits are the page number and the rightmost *m* bits are the offset. From the example n=6 and m=10.
- The following steps are needed for address translation:

  1. Extract the page number as the leftmost n bits of the logical address.

  2. Use the page number as an index into the process page table to find the frame number *k*.

Paging

3. The starting physical address of the frame is
   - k X 2m, and the physical address of the referenced byte is that number plus the offset.
   - This physical address need not be calculated; it is easily constructed by appending the frame number to the offset.

- From our example the logical address is 0000010111011110, which page number 1 and offset 478.

Paging

- Suppose that this page is residing in memory frame 6 = binary 000110.
- Then the physical address is frame number 6, offset 478 = 0001100111011110 (Fig 7 12a)

Paging

- Summarize for simple paging:
  - Memory is divided into small equal-size frames.
  - Each process is divided into frame-size pages
  - Smaller processes require fewer pages larger pages, processes require more.
  - When a process is brought in, all of its pages are loaded into available frames, and a page table is set up.
  - This approach solves many problems inherentin partitioning.

SEGMENTATION

Segmentation

- Alternative way is to subdivided user program into segments.
- All segments of all programs do not have to be of the same length, but there is a maximum length.
- Addressing consist of two parts - a segment number and an offset
- Since segments are not equal, segmentation is similar to dynamic partitioning the different is that with segmentation a program may occupy more than one partition and these partitions need not be contiguous.

Segmentation

- Segmentation eliminates internal fragmentation but suffer from external fragmentation.
- However because a process is broken up into a number of smaller pieces, the external fragmentation should be less.
- Paging is invisible for the programmer but segmentation is visible and is provided as a convenience for organizing programs and data.

Segmentation

- Programmer or compiler will assign programs and data to different segments
- For modular programming, the program or data may be further broken down into multiple segments

Segmentation

- The principal inconvenience is that the programmer must be aware of the maximum segment size limitation.
- For unequal size segments there is no simple relationship between logical addresses and physical addresses.
- Analogous form paging, a simple segmentation scheme would make use of segment table for each process and a list of free blocks of memory.

Segmentation

- Each segment table entry would have to give the starting address in memory of the corresponding segment.
- The entry should also provide the length of the segment, to assure that invalid addresses are not used.
- When process enters the Running state, the address of its segment table is loaded into a special register used by the memory management hardware.

Segmentation

- Consider and address of *n* + *m* bits, where the leftmost *n* bits are segment number and the rightmost *m* bits are the offset.
- In our example (Fig. 7.11c), n=4 and m=12. Thus
- the maximum segment size is 212 = 4096.
- The following steps are needed for address translation:
  1. Extract the segment number as the leftmost n bits of the logical address.
  2. Use the segment number as an index into the process segment table to find the starting physical address of the segment.

Segmentation

3. Compare the offset, expressed in the rightmost m bits, to the length of the segment. If the offset greater than the length, the address is invalid.
4. The desired physical address is the sum of the starting physical address of the segment plus the offset.

- Example the logical address 0001001011110000, which is segment number 1 (0001) and offset 75 (001011110000).

Segmentation

- Suppose that this segment is residing in memory starting at physical address 0010000000100000.
- Then the physical address is (Fig 7.12b)

0100000000100000 + 001011110000

= 0010001100010000.

Segmentation

- Summarization for simple segmentation:
  - A process is divided into a number of segments which need not be equal size.
  - When a process brought in, all of its segments are loaded into available regions of memory, and a segment table is setup.