Mohamed Kheider Biskra University
Faculty of Exact Sciences and Natural and Life Sciences
Department of Computer Science

Course material

Module: Operating Systems I

Chapter 2: Memory Management

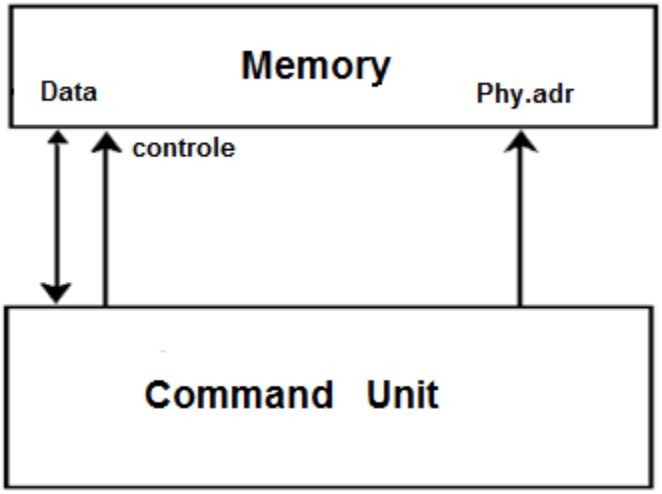
(Part 01)

Level: 2LMD
Module Leader: Dr. D. Boukhlouf

Academic year 2023/2024

1. Introduction

Memory is an organ whose function is to store and retrieve binary words identified by an address. It communicates with the rest of the computer via 3 buses:
- the **control bus** (command bus), which indicates to the memory the operation you want to perform
- the bidirectional **data bus**, which is used to send and receive words;
- the **address bus**, which indicates the address in question to memory.



Memory is an important resource that needs to be carefully managed. It is capable of storing:

- Passive objects as file directories (programs, images, text, and sound), I/O data buffers, or network transfer buffers.
- Active objects such as processes.

It is divided into a finite suite of components called slots. A program can only run if its instructions and data (at least partially) are in main memory. If you want to run several programs at the same time; each one needs to be loaded into the main memory.

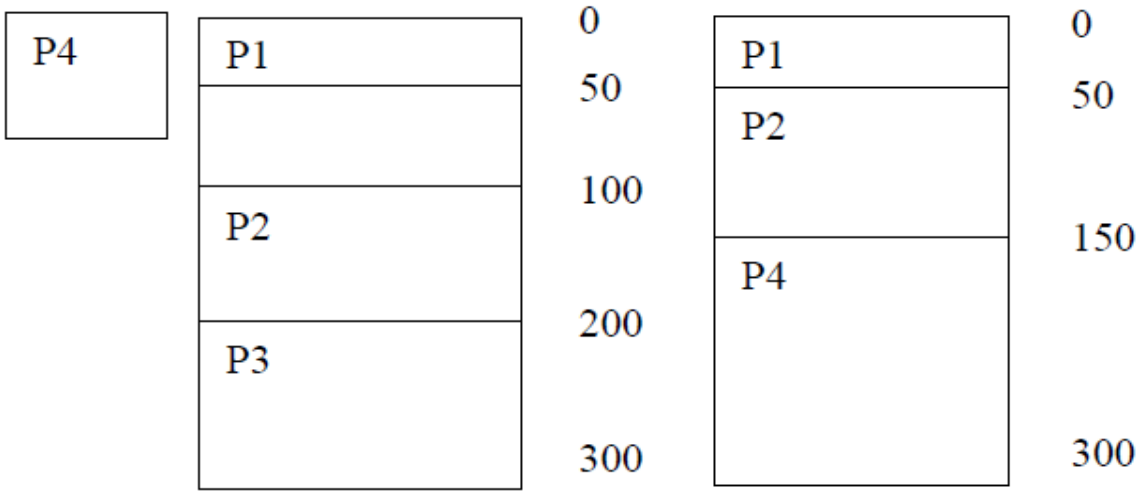
2. ***Memory manager***: A memory manager must perform the following functions:
1. Keep track of the status of each portion of the memory (free or allocated).
2. Determine a memory allocation policy that allows you to decide who should have memory, how much, and for how long.
3. Determine an allocation technique that allocates space to a process.
4. Determine a memory release policy.

The goal of good memory management is to increase the overall efficiency of the system. To do this, there must be enough processes (tasks) in memory to ensure a good distribution between I/O and CPU demand. It should have the following objectives:

2.1 Reallocation: The operating system allocates a memory area to each program. But not all programs are the same size. In addition, programs are launched by users and then terminate at those times that the system does not know in advance. Each time a user requests the launch of a program, the system has to find a place in the memory to load it: there is a reorganization of programs into memory.

Example:

Let the programs P1, P2 and P3 be in memory as shown in the following figure. If a fourth program of size 150 were to be executed, this would obviously be impossible. Now, suppose the P3 program terminates, the total available space in memory is 150. Therefore, the P4 program can theoretically be loaded.
Unfortunately, in our example, this will still be impossible, because the 150 available units do not form a contiguous space in which to load the P4 program. A rearrangement of the memory space will load P4 as follows:



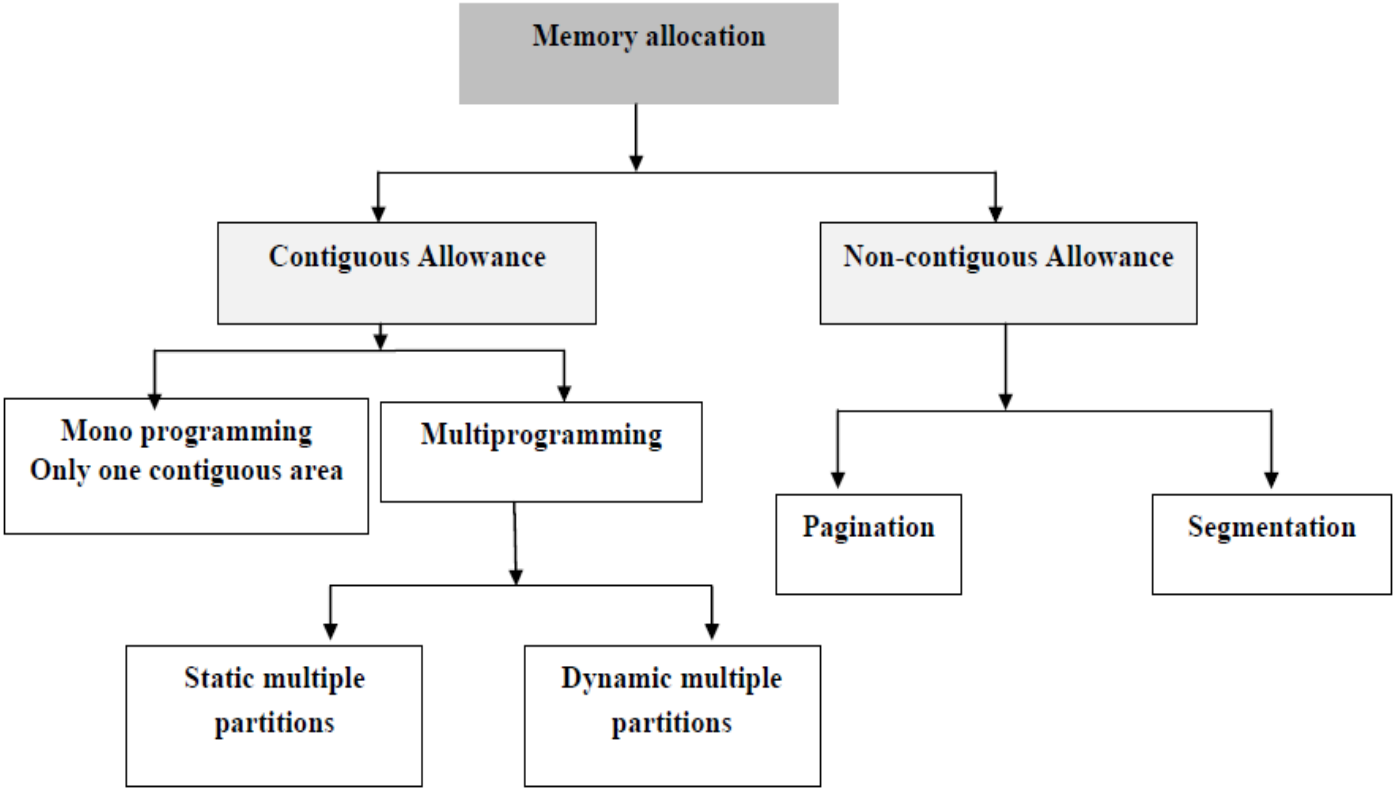
2.2 Protection: The coexistence of several processes in central memory requires the protection of each memory space from the others. Therefore, a P1 process can only access the space of a P2 process if it is allowed.

2.3 Sharing: Sometimes it's useful to share memory space between multiple processes. Thus, the memory management subsystem must allow controlled access without compromising protection.

Example: A text editor shared between multiple users of a timeshare system.

3. Memory Allocation Strategies:

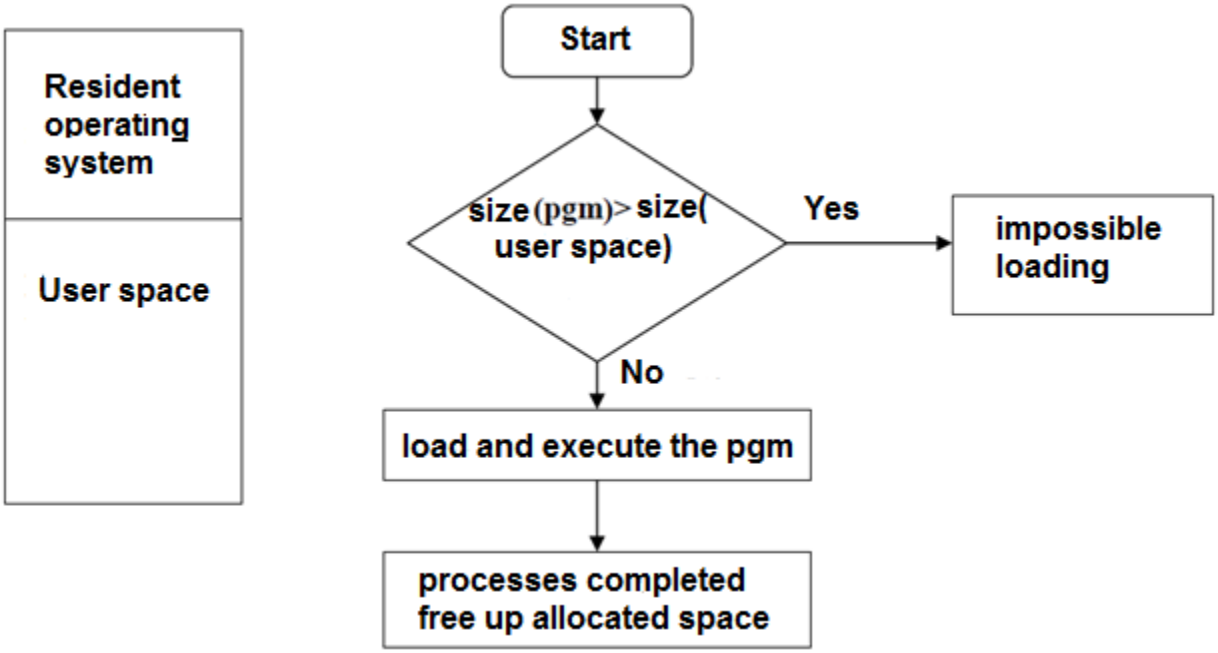
There are basically two modes of main memory management: contiguous and non-contiguous. Depending on the memory management mode that is applied, when a program is loaded into main memory from disk, the program will be placed in a single zone (contiguous allocation) or distributed among multiple zones (non-contiguous allocation).



3.1 Contiguous Allocation

3.1.1 A single contiguous zone (mono-programming):

In this case, the memory is divided into two contiguous areas. One is permanently allocated to the resident part of the operating system. The second zone will allow you to relocate a user process or a non-resident system. This strategy is used by PC/DOS microcomputers.
Memory management is simple; the operating system must keep track of two memory areas. The allocation algorithm can be described by the following flowchart:



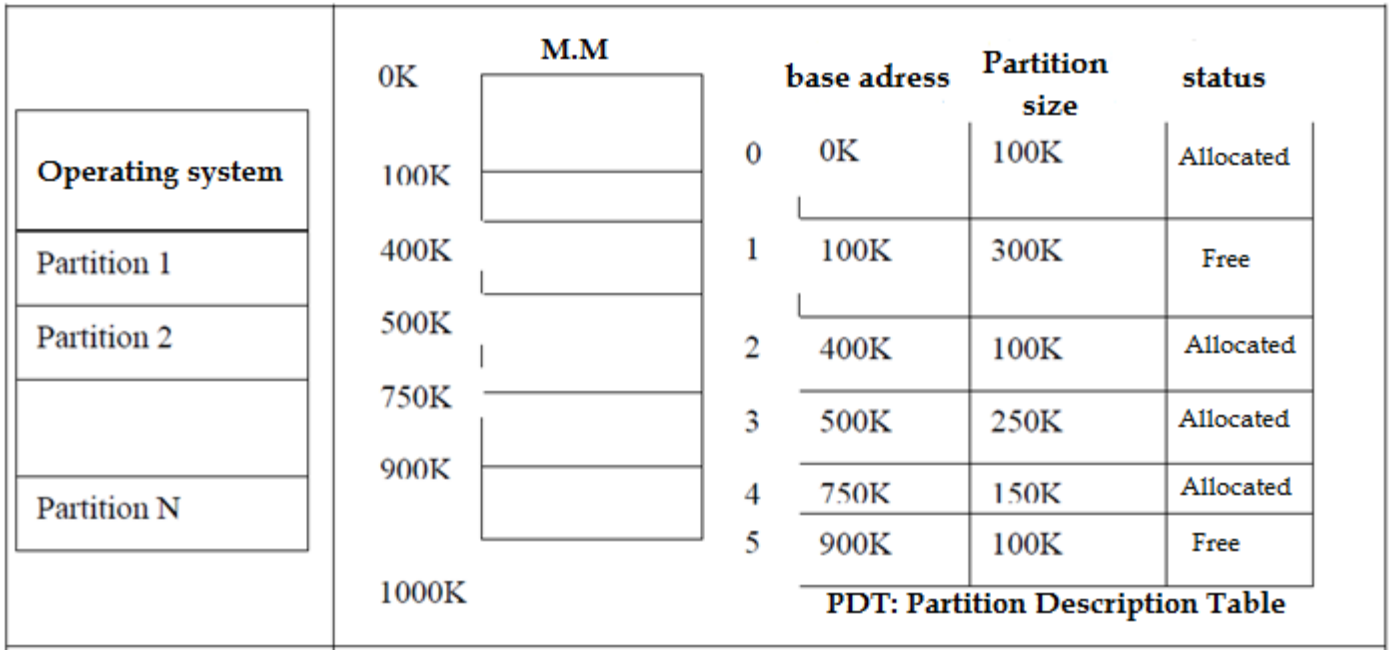
The major disadvantage of this strategy is the misuse of memory space in the sense that programs (usually) partially occupy user space. In addition, the size of user programs is limited by the size of the user space.

3.1.2 Multiple partitions (multiprogramming)

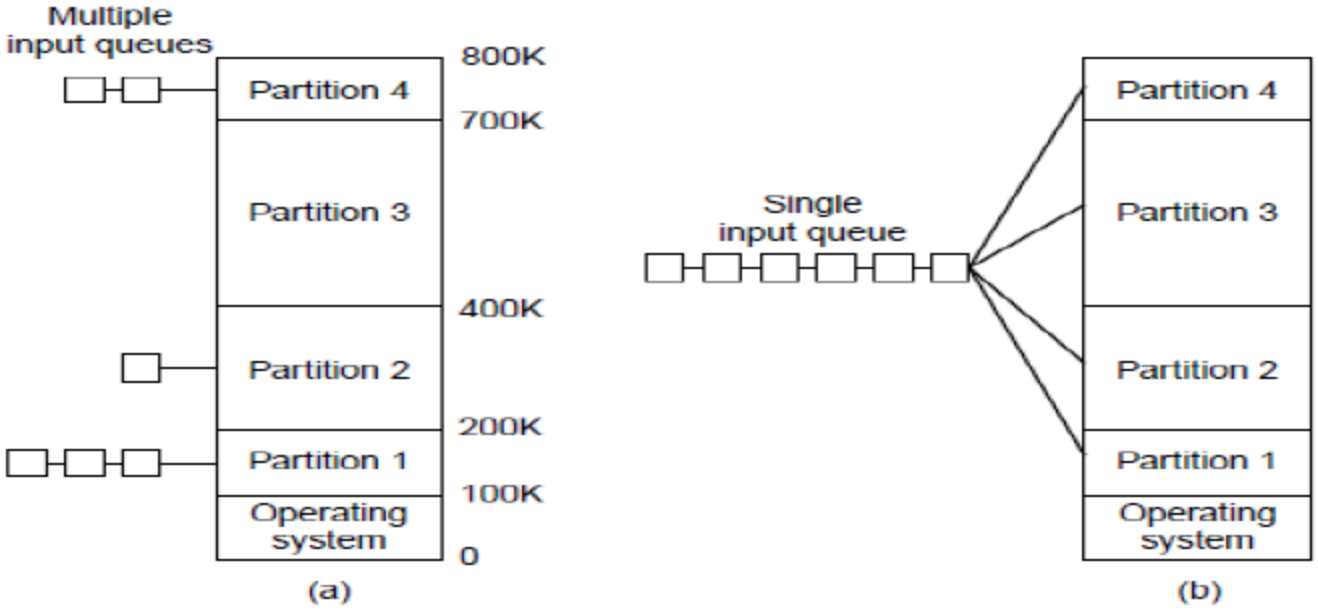
Most modern OSs allows multiple processes to run at the same time: When one process gets stuck waiting for an I/O, another can use the CPU.

3.1.2.1 Static Multiple Partitions

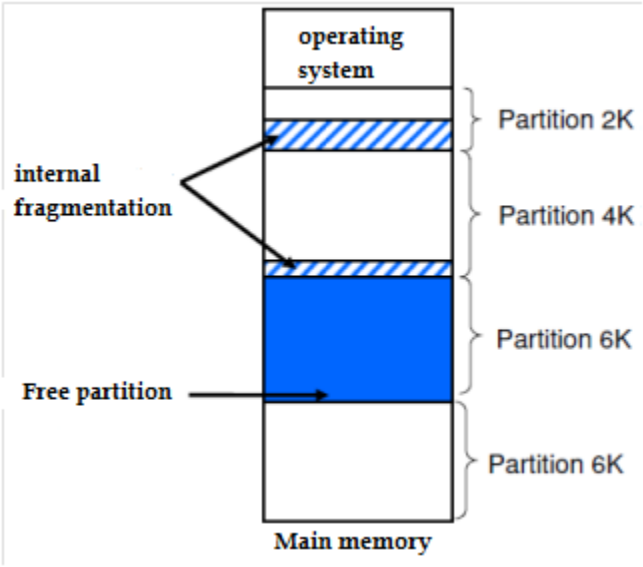
The simplest way of multiprogramming is to subdivide memory into n fixed-size partitions. The number and size of each partition is set at the time the system is generated. Each partition can contain exactly one process. The memory manager maintains a table that shows which parts of the memory are available and which are occupied: a Partition Description Table (PDT).



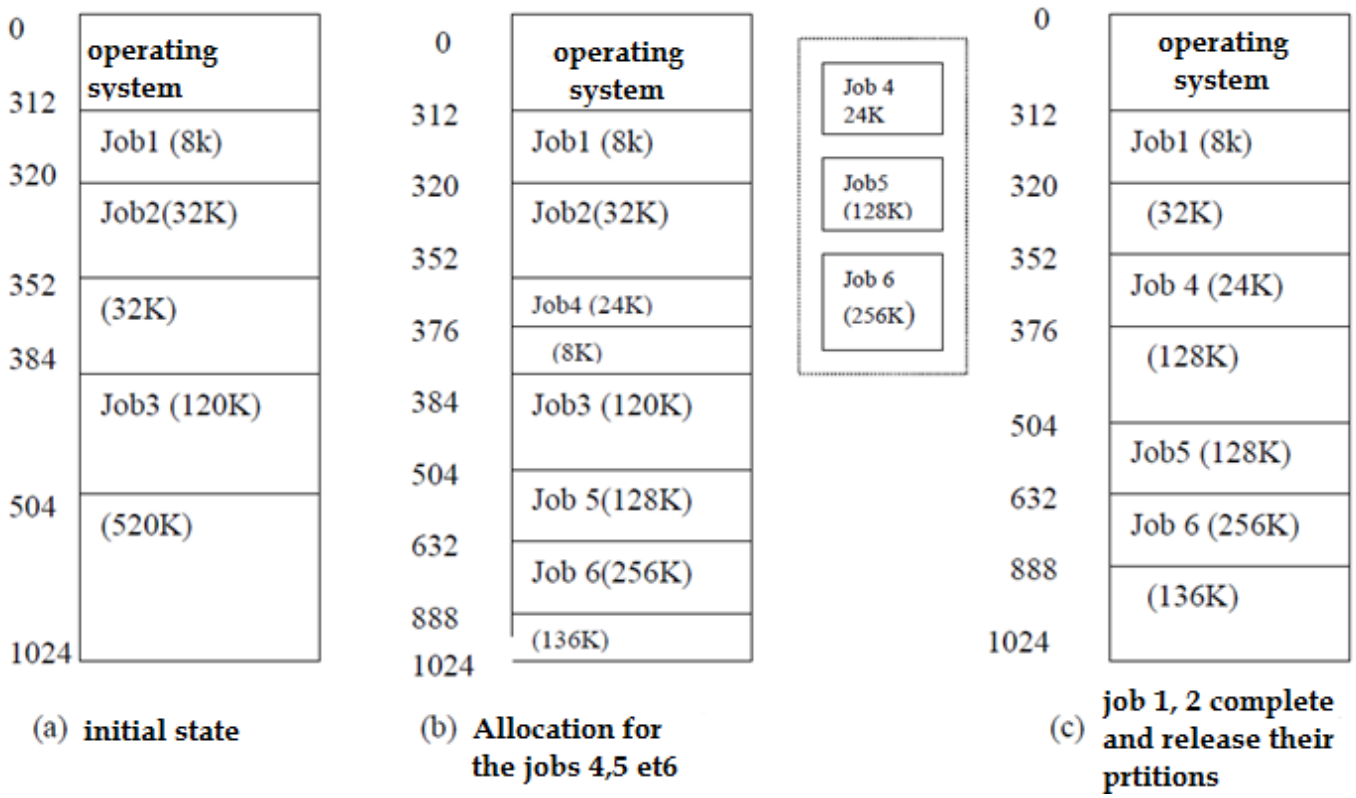
There are two management methods:
- <u>One queue is created per partition</u>. Each new process is queued in the smallest partition that can hold it.
Disadvantages:
* We usually lose space within each score
* there may be unused partitions (their queue is empty)
<u>- Create a single global queue. There are two strategies:</u>
* As soon as a partition becomes available, it is assigned the first task in the queue that can fit in it.
Disadvantage: This allows you to assign a large partition to a small task and waste a lot of space
* As soon as a partition becomes available, it is assigned the biggest task in the queue that can fit in it. The disadvantage is that small processes are penalized.



Internal fragmentation:
The allocated memory may be larger than the required memory. This difference is internal to a partition but is not used.
External fragmentation:
When there is not enough partition to accommodate a process, then the sum of the free partitions allows the process to load.



3.1.2.2 Variable multiple partitions:
In order to improve the fixed-partition memory management strategy, memory will be partitioned dynamically according to demand. Each program is allocated a partition exactly equal to its size. When a program finishes executing, its partition is retrieved by the system to be allocated to another program completely or partially depending on the request.
Example:
That is a memory of 1024 KB of which 312 KB is occupied by the OS. At time T1, we have the configuration according to the figure below.
In a variable-partition organization, programs are placed in partitions according to different strategies. To do this, the memory manager needs to keep track of occupied partitions and free partitions. One solution is to maintain two tables, one for occupied partitions and one for free partitions.



Placement Strategies:

In a variable-partition organization, the placement of programs in partitions is done according to a number of possible strategies. It should be noted that there are a number of possible strategies. It should be noted that the behavior of a strategy depends very much on the nature of the requests:

a) **The first fit strategy:** In this strategy, the table of free partitions is sorted in order of increasing addresses. For the allocation of a given partition, we will start with the free partition with the lowest address and the search continues until we encounter the first partition whose size is at least equal to that of the waiting program.

b) **The best fit strategy:** In this case, the table of free partitions is sorted by increasing sizes. For the allocation of a given partition, we will start with the smallest free partition. The search continues until the first partition is found that is at least the size of the pending program.

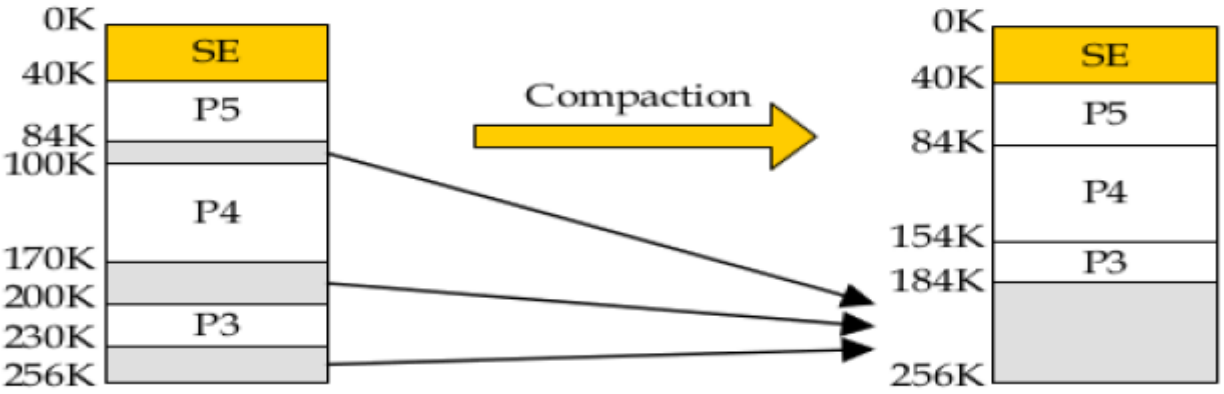
c) **The Worst Fit Strategy:** In this case, the job is assigned the largest partition. You have to go through the whole table unless it's sorted.

Remark

The other two strategies are cumbersome to implement; there is a problem of external fragmentation and the solution is compaction.

Compaction:

Compaction is a solution for external fragmentation that allows unused space to be grouped together. The compaction operation is performed when a program that asks to be run does not find a partition large enough, but its size is smaller than the existing external fragmentation

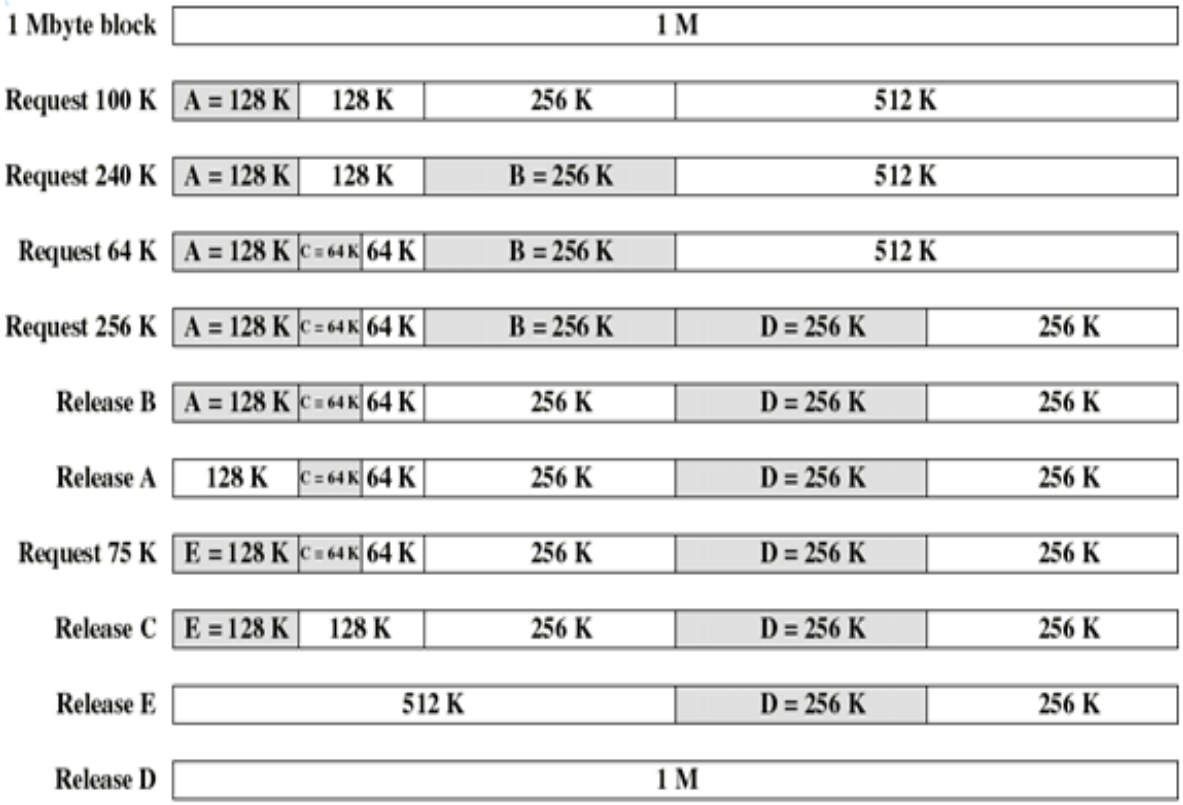


3.1.2.2 Buddy System

A reasonable compromise to overcome disadvantages of both fixed and variable partitioning schemes.
We start with the entire block of size 2^{U}
When a request of size S is made:

- If 2^{U-1} < S <= 2^{U} then allocate the entire block of size 2^{U}
- Else, split this block into two buddies, each of size 2^{U-1}
- If 2^{U-2} < S <= 2^{U-1} then allocate one of the 2 buddies
- Otherwise one of the 2 buddies is split again

This process is repeated until the smallest block greater or equal to S is generated
Two buddies are coalesced whenever both of them become unallocated
Example:



Advantages

- In comparison to other simpler techniques, the buddy memory system has little external fragmentation.
- The buddy memory allocation system is implemented using a binary tree to represent used or unused split memory blocks.
- Allocates a block of the correct size.

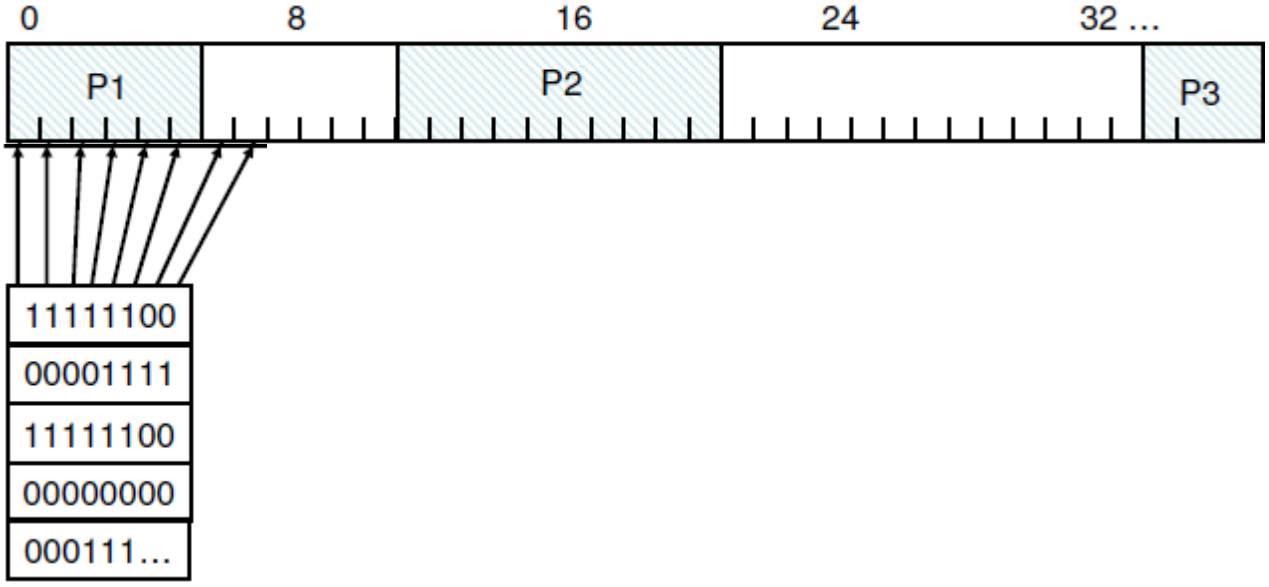
- The buddy system is very fast to allocate or deallocate memory.
- In buddy systems, the cost to allocate and free a block of memory is low compared to that of best-fit or first-fit algorithms.
- It is easy to merge adjacent holes.

Disadvantages

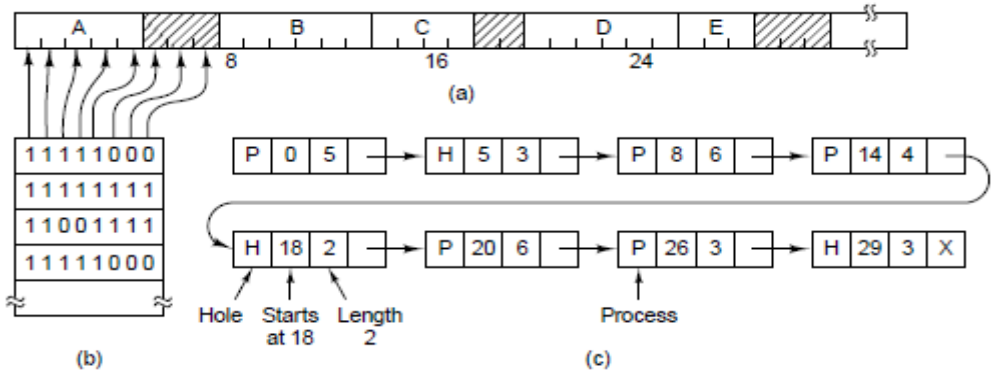
- It requires all allocation units to be powers of two.
- It leads to internal fragmentation.

State of Memory

To manage the allocation and release of memory space, the manager needs to know the state of the memory:
a) Bit table: The state of memory blocks can be preserved using a bit table. Free units are denoted by 0, those occupied by a 1 (or vice versa).



Fundamental element in the configuration:
When the allocation unit is **larger, the bit array is smaller**, but a **significant** amount of memory can be wasted in the last unit allocated to a process with a size that is not a multiple of the allocation unit.
Bit map advantage: Simple way to keep track of memory words in a fixed amount of memory Bit map disadvantage: When a process of k units is loaded into memory, the memory manager must scan the *bit map* to find a sequence of k consecutive bits with a value of 0. **This search can be slow**.
Maintain a linked list of allocated and free heaps. In this list, an element is either a process or a gap between two processes.



b) Linked lists: Memory can be represented by a chained list of structures whose members are: type (free or busy), start address, length, and a pointer to the next item. When a process is completed or transferred to disk, it takes time (updating the linked lists) to examine whether a grouping with its neighbors is possible to avoid excessive memory fragmentation.

In summary, chained lists are a faster solution than the previous one for allocation, but slower for release.

c) Releasing a Partition:
There are three cases to consider when releasing a partition:

- The freed partition is surrounded by two free partitions.
- The freed partition is surrounded by a free partition and an occupied partition.
- The freed partition is surrounded by two occupied partitions.

4. The back and forth:
Back-and-forth is implemented when not all processes can be held in memory at the same time. Some of them must then be temporarily moved to a memory, usually a reserved part of the disk (swap area or baking store).
On disk, the reciprocating area of a process can be allocated to the request in the general swap area. When a process is unloaded from central memory, a place is placed in it. Back-and-forth places are generated in the same way as the central memory. The back-and-forth area of a process can also be allocated once and for all at the start of execution. When unloading, the process is sure to have a free waiting area on the disk.
The system runs processes in temporary memory for a certain amount of time. The replacement algorithm can be the turnstile.

Logical or physical address space

The CPU manipulates **logical addresses** (relative location). Programs only know logical or virtual addresses. The logical (virtual) address space is therefore a set of addresses that can be generated by a program.
The memory unit manipulates **physical addresses** (memory location). They are never seen by user programs. The physical address space is a set of physical addresses corresponding to a logical address space.