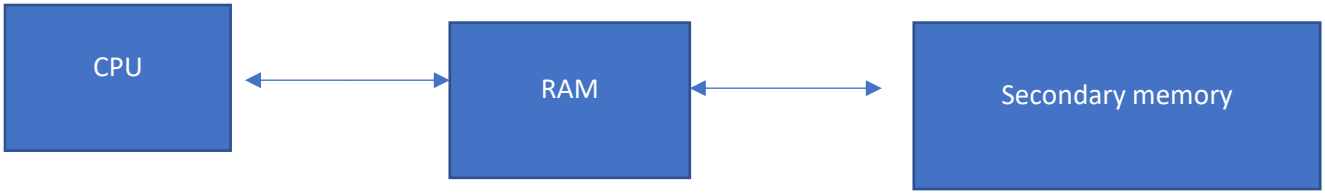
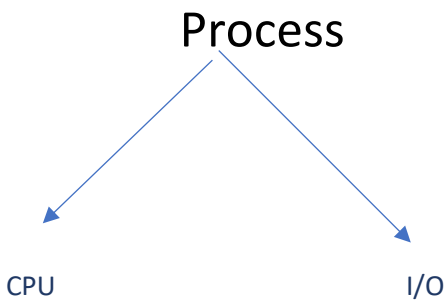
Memory management

Memory is a large array of bytes. Every byte is having some addresses.

Memory management is the kind of method to manage the primary memory. The goal of memory management is utilizing the memory efficiently.



The purpose of secondary memory is to execute the program. The program is nothing but some instructions. Secondary memory is slower than CPU so we cannot connect them directly.



Every process is either executing on CPU or requesting for I/O. We need to swap all the processes.

When process is requesting for I/O then CPU release the process & another process is coming from the ram and executing on the CPU.

Let, ram size = 4MB.

Process size = 4MB

Number of process = 4MB/ 4MB = 1

One process can come from secondary memory to primary memory (ram) at a time.

Another example,

Let, ram size = 8MB.

Process size = 4MB

Number of process = 8MB/ 4MB = 2

Two processes can come from secondary memory to primary memory (ram) at a time .

CPU Utilization:

Let,

K = time for I/O operations for p1 process = 0.7 = 70%

Total Execution time = 1

CPU Utilizing = (1-k) = .3=30%

Another example for p1 &p2 processes,

K^2 = time for I/O operations = $(0.7)^2$ = .49 = 49%

CPU Utilizing = (1- K^2) = 0.51 = 51%

If the size of ram is increased then Number of process is increased. Finally, CPU utilization is increased.

Types of memory management

There are 2 types of memory management processes. Such as –

1) Contiguous:

    a) Fixed partition (Static)

    b) Variable partition (Dynamic)

2) Non Contiguous:

    a) Paging

    b) Multilevel paging

    c) Inverted paging

    d) Segmentation

    e) Segmentation Paging

Note: Operating system mounted on ram and collapsed some memory spaces.

Fixed partitioning

Partitioning the ram before allocating the process.

- Number of partitions are fixed
- Size of each partition may or may not same.

OS	Some spaces
P1	4MB
P2	6MB
P3	8MB

OS	Some spaces
P1	6MB
P2	6MB
P3	6MB

Advantages:

1. Easy to implement.
2. Allocation and deallocation processes are also easy.

Disadvantages:

1. Internal fragmentation:

OS
8MB → 7MB P1
4MB

When the partition size of ram is bigger than the process size then Internal fragmentation occurs. Here, partition size of ram is 8 MB, process size is 7MB. So , 1MB space is wasted. This event is called internal fragmentation.

2. Limit Of process size:
   Maximum size of partition < process size .
   Process never fits into the ram if the Maximum size of partition is less than the process size.
3. Limitation on degree of Multiprogramming:
   The number of ram's slot is fixed . As many programs as we want can not be allocated into ram.
4. External fragmentation:
   If there is a internal fragmentation , there will be a external fragmentation.

OS
P1(7MB) ⟹ 8MB
P2(3MB) ⟹ 8MB

KYAU,OS

P3(12MB) ⟹ 16MB

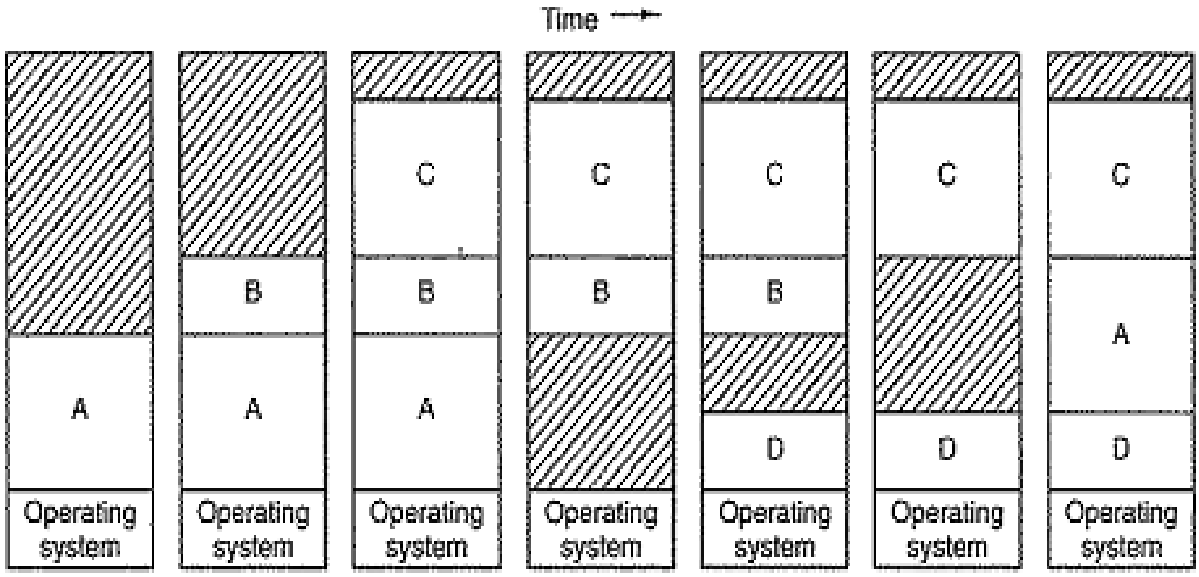
Internal fragmentation is occurred . Total empty space is (1+5+4) MB = 10MB. Now , process p4(6MB) cannot allocate into the ram even there is enough empty space. This occurrence is called external fragmentation.

Dynamic Partitioning

Partitioning the ram when allocating the process . According to the need we divide the memory at run time.

Swapping

The operation of a swapping system is illustrated in Fig.



Initially, only process A is in memory. Then processes B and C are created or swapped in from disk. Then A is swapped out to disk. Then D comes in and B goes out. Finally A comes in again. Since A is now at a different location, addresses contained in it must be relocated, either by software when it is swapped in or (more likely) by hardware during program execution.

When swapping creates multiple holes in memory, it is possible to combine them all into one big one by moving all the processes downward as far as possible. This technique is known as **memory compaction**.

When memory is assigned dynamically, the operating system must manage it. In general terms, there are two ways to keep track of memory usage:

Bitmaps and

Linked lists.

Advantages:

I. No internal fragmentation
II. No limitation on number of process
III. No limit of process size.

Disadvantages:

- Allocation and deallocation is complex.
- External fragmentation:

OS
P1(4mb) ⟶ 4mb
P2(4mb) ⟶ 4mb
P3(4mb) ⟶ 4mb

Now the processes p1 & p3 are completed. New process p4(6Mb) cannot enter the memory even the memory is empty.

Managing Free Memory

When the processes and holes are kept on a list sorted by address, several algorithms can be used to allocate memory for a created process (or an existing process being swapped in from disk). We assume that the memory manager knows how much memory to allocate.

The simplest algorithm is **first fit**. The memory manager scans along the list of segments until it finds a hole that is big enough. The hole is then broken up into two pieces, one for the process and one for the unused memory. First fit is a fast algorithm because it searches as little as possible.

A minor variation of first fit is **next fit**. It works the same way as first fit, except that it keeps track of where it is whenever it finds a suitable hole. The next time it is called to find a hole, it starts searching the list from the place where it left off last time, instead of always at the beginning, as first fit does. Simulations by Bays (1977) show that next fit gives slightly worse performance than first fit.

Another well-known and widely used algorithm is **best fit**. Best fit searches the entire list, from beginning to end, and takes the smallest hole that is adequate.

Rather than breaking up a big hole that might be needed later, best fit tries to find a hole that is close to the actual size needed, to best match the request and the available holes.

Best fit is slower than first fit because it must search the entire list every time it is called. Somewhat surprisingly, it also results in more wasted memory than first fit or next fit because it tends to fill up memory with tiny, useless holes. First fit generates larger holes on the average.

To get around the problem of breaking up nearly exact matches into a process and a tiny hole, one could think about **worst fit**, that is, always take the largest available hole, so that the new hole will be big enough to be useful. Simulation has shown that worst fit is not a very good idea either..

At a glance:

empty	25k
P1	20k
empty	40k
P2	40k

First fit: Allocate the first hole that is big enough.

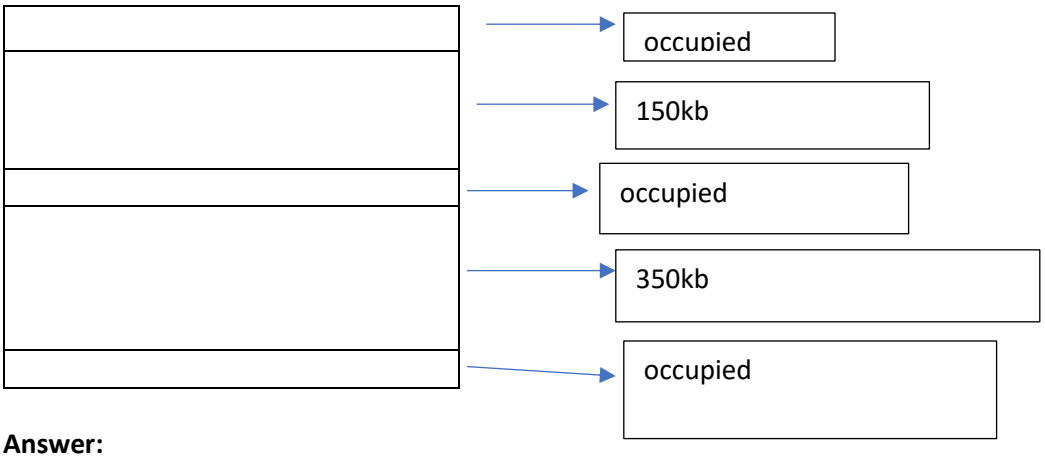
Next fit: same as first fit but start search always from last allocated hole.

Best fit: Allocate the smallest hole that is big enough(Finding minimum internal fragmentation)

Worst fit: Allocate the largest hole.

Question: Requests from processes are 300kb,25kb,125kb, 50kb respectively the above request could be satisfy with ----------

A. Best fit but not first fit
B. First fit but not best fit
C. First fit and worst fit



Answer:

Let,

300kb = p1,

25kb = p2,

125kb = p3,

50kb = p4

First Fit:

occupied	**occupied**
150kb	**P2**
	P3
occupied	**occupied**
150kb	**P1**
	P4
occupied	**occupied**

Best Fit:

occupied	**occupied**
150kb	**P3**
	Hole(25k remaining)
occupied	**occupied**
150kb	**P1**
	P2
	Hole(25k remaining)
occupied	**occupied**

Here external fragmentation occurred. We cannot divide the process p4 to fit into memory even there is enough available space.

Worst fit:

Try yourself……..