SPOS-05

① Bitmap:

Bitmap is the least famous data structure to store the details. In this scheme, the main memory is divided into collection of allocation units. One or more allocation units may be allocated to a process. However, the size of the allocation unit is fixed that is defined by operating system and never changed. Although the partition size may vary but the allocation size is fixed.

The main task of the operating system is to keep track of whether the partition is free or filled. For this purpose, the operating system also manages another data structure that is called bitmap.

2) Disadvantages of bitmap:

The OS has to assign some memory for bitmap as well since it stores the details about allocation units. That much amount of memory cannot be used to load any process therefore that decreases the degree of mutiprogramming as well as throughput.

The allocation unit is of 4 bits that is 0.5 bits. Here, 1bit of the bitmap is representing 1 bit of allocation unit.

size of 1 allocation unit = 4bits

size of bitmap configuration = $1/(4+1)$

$= \frac{1}{5}$ of total main memory

$\therefore$ In this bitmap configuration, $\frac{1}{5}$ of total main memory is wasted.

To identify any hole in the memory, the OS need to search the string of 0s in the bitmap. This searching tasks is a huge amount of time which makes the system inefficient to some extent.

3) How linked list is used for dynamic partioning?

The most popular approach to keep track the free or filled partition is used linked list.

The OS maintains a linked list where node represents each partitions. every node has 3 fields

1. The node stores a flag bit which shows whether the partition is a hole or some process inside.

2. Second filled stores starting inden of the partition.

3. Third field stores end index of partition while using this approach,

*The OS must be very clear about the location of new code which is to be added in linked list. The node should be added in linked list. The node should be added in increasing order of starting index.

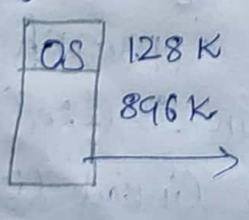
5. partitioning algorithm:

1. First fit algorithm:

It scans the linked list and whenever it finds the first big enogh hole to store a process, it stops scanning and load the process into that hole. This procedure produces 2 partition.

out of them, one partition will be a hole while the other will store process. If maintains the linked list according to the increasing order of starting index.

Eg.



the complete process is stored here.

2. Next fit algorithm:

* it is similar to first bit except the fact that, next fit scans the linked list from the node where it previously allocated a hole. It doesn't scan the whole list

it starts scanning from the next node,

3. Best fit algorithm:

* It tries to findout the smallest hole possible in the list that can accomodate the size requirement of process.

4. worst fit algorithm:

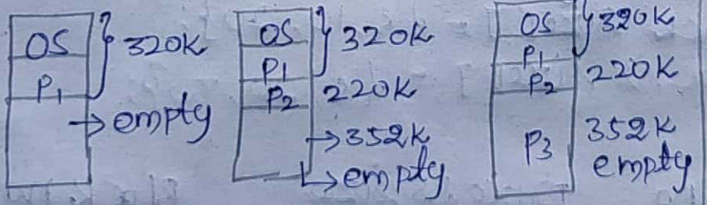
* It scans the entire list every time and tires to find out the biggest hole in the list. which can fulfil the list of the process.

* This algorithm produces the large holes to load the other processes, this is not better approach due to the fact that it is slower because it searches entire list every time.

5. quick fit algorithm:

* The quick fit algorithms maintains the different lists of frequently used sizes. Although, it is not practicall suggestible because the procedure takes so much time to create different lists and then expending the holes to load a process.

Eg



according to the process arriving the memory gets divided according to the process.

i) First fit algorithm:

eg: Input: blocksize[] = [100, 300, 200, 500, 600];
process size[] = {212, 417, 112, 426};

output:

process no	process size	Block no
1	212	2
2	417	5
3	112	2
4	426	not allocated.

2. Next fit algorithm:

eg: Input: block size[] = {5,10,20}
process size [] = {10,20,30};

output:

process no	process size	Block no
1	10	2
2	20	3
3	30	not allocated.

3. Best fit algorithm:

eg: Input: block size[] = {100, 300, 200, 500, 600};
process size[] = {212, 417, 112, 426};

output:

process no	process size	Block no
1	212	4
2	417	2
3	112	3
4	426	5

4. worst fit algorithm:

Block size[] = {100, 500, 200, 300, 600};
process size[] = {212, 417, 112, 426};

output:

process no	process size	Block no
1	212	5
2	417	2
3	112	5
4	426	not allocated.

5. quick fit algorithm:

Block size[] = {5, 10, 20};
process size[] = {5, 20, 30};

output:

process no	process size	Block no
1	10	2
2	20	3
3	30	not allocated.