Memory Management (1)

Function

- Keep track of which parts of memory are in use and which parts are not in use
- Manages memory allocation and deallocation
- Manages swapping

Background

Memory management related to main memory as a resource that must be allocated and used together by processes.

Main memory must be managed carefully :

- Because of the limited capacity
- Program must be brought into memory and placed within a process for it to be executed.

Swapping

A process can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution.

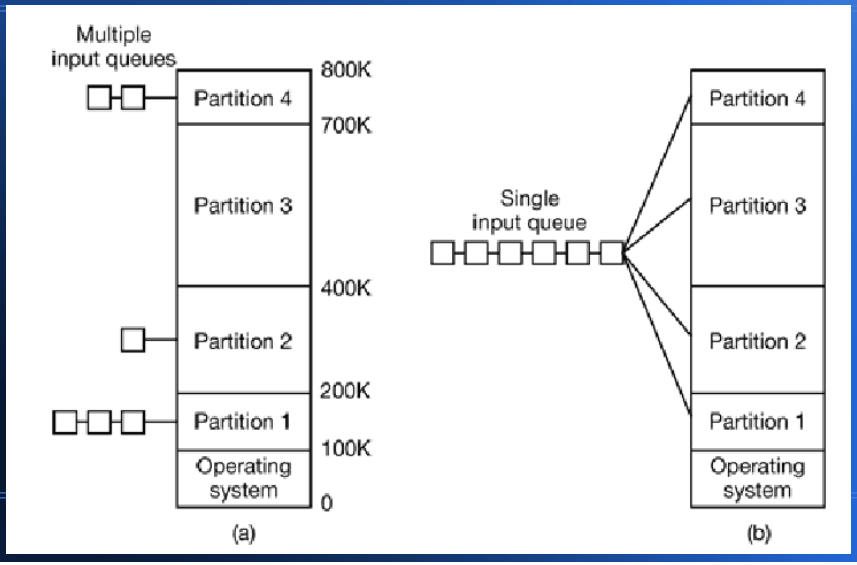
Memory management systems can be divided into 2 :

- Without swapping
- With swapping

Monoprogramming Without Swapping

- Run just 1 program at a time
- All resources are controlled by an active process.
- Memory allocation between the process is done in sequence

Multiprogramming With Fixed Partition



Multiprogramming With Fixed Partition

- Multiple processes run at the same time
- Having multiple processes running at once means that when one process is blocked waiting for I/O to finish, another one can use the CPU
- The easiest way to achieve multiprogramming is simply to divide memory up into n (possibly unequal) partitions.

Fixed Partition

- Memory is divided into several fixed partition.
- Fixed partition can be distinguished based on the size :
  - Same size partition
  - Different size partition

Same Size Partition

Disadvantage :

- If the program size is bigger than the partition size – program cannot be executed – external fragmentation, solution : overlay (divide the program into parts that can be loaded to memory)
- It the program size is a lot smaller than the partition size – wasted memory space – internal fragmentation, solution : different size partition

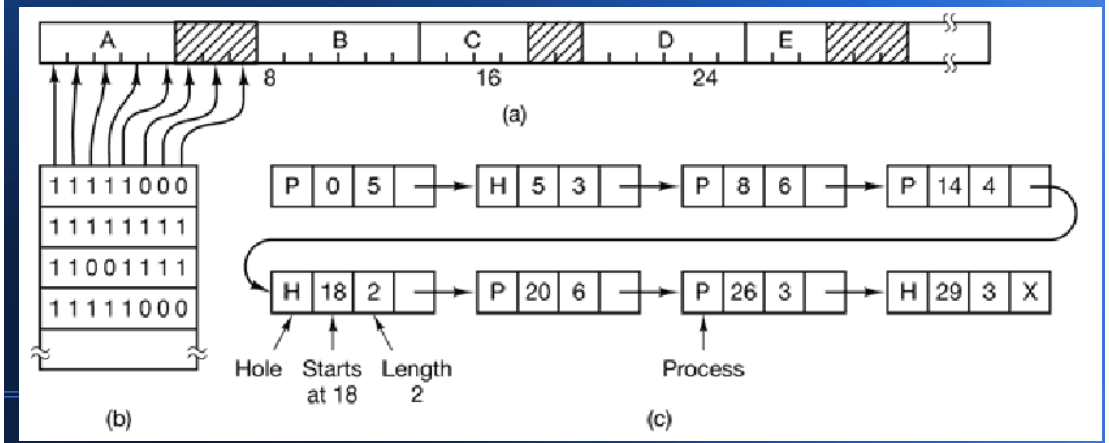
Dynamic Partition

- The partition size could change dynamically according to the process size.
- Problem :
  - Holes between partitions, solution : memory compaction
  - Growing process, solution :
    - Use the next empty partition
    - Move the process to the larger partition
    - Swapping
    - Stop the process

Different Size Partition

Memory is divided into parts that have different size

there are two ways to keep track of memory usage (a) : bitmaps (b) and free Lists (c)



Memory Allocation Algorithms

- several algorithms can be used to allocate memory for a newly created process (or an existing process being swapped in from disk).
- assume that the memory manager knows how much memory to allocate.

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.

First Fit

- The memory manager scans along the list of segments until it finds a hole that is big enough.
- First fit is a fast algorithm because it searches as little as possible.

Best Fit

- Best fit searches the entire list 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.

Worst-fit

- always take the largest available hole, so that the hole broken off will be big enough to be useful.

Buddy System

- Specify upper limit (u) dan bottom limit (l)
- Example : 2000 kb memory, upper limit value is 20 because 2^{20} = 1024, the rest 976kb will be allocated on smaller block. Bottom limit usually around 12-16, if too small will burden the system because it must remember too many allocated and unallocated, for example the bottom limit is 16, so 2^{16} = 64k

Buddy System

- is a memory allocation algorithm that divides memory into partitions to try to satisfy a memory request as suitably as possible. This system makes use of splitting memory into halves to try to give a best-fit
- Every memory block in this system has an *order*, where the order is an integer ranging from 0 to a specified upper limit.
- The blocks in each order have sizes proportional to 2^{order}, so that each block is exactly twice the size of blocks that are one order lower.

Buddy System

Contoh :

1=Program A requests 34k

2=Program B requests 66k

3=Program C requests 35k

4=Program D requests 67k

5=Program C releases its memory

6=Program A releases its memory

7=Program B releases its memory

8=Program D releases its memory

Buddy System Steps

1. If memory is to be allocated

Look for a memory slot of a suitable size (the minimal 2^k block that is larger or equal to that of the requested memory)

- If it is found, it is allocated to the program
- If not, it tries to make a suitable memory slot. The system does so by trying the following:
  - Split a free memory slot larger than the requested memory size into half
  - If the lower limit is reached, then allocate that amount of memory
  - Go back to step 1 (look for a memory slot of a suitable size)
  - Repeat this process until a suitable memory slot is found

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t = 0$	1024K															
$t = 1$	A-64K	64K	128K		256K				512K							
$t = 2$	A-64K	64K	B-128K		256K				512K							
$t = 3$	A-64K	C-64K	B-128K		256K				512K							
$t = 4$	A-64K	C-64K	B-128K		D-128K		128K		512K							
$t = 5$	A-64K	64K	B-128K		D-128K		128K		512K							
$t = 6$	128K		B-128K		D-128K		128K		512K							
$t = 7$	256K				D-128K		128K		512K							
$t = 8$	1024K															

1=Program A requests 34k
2=Program B requests 66k
3=Program C requests 35k
4=Program D requests 67k
5=Program C releases its memory
6=Program A releases its memory
7=Program B releases its memory
8=Program D releases its memory

Buddy System Steps

2. If memory is to be freed

Free the block of memory

Look at the neighboring block - is it free too?

If it is, combine the two, and go back to step 2 and repeat this process until either the upper limit is reached (all memory is freed), or until a non-free neighbour block is encountered