Friday, April 1, 2022 10:39 AM

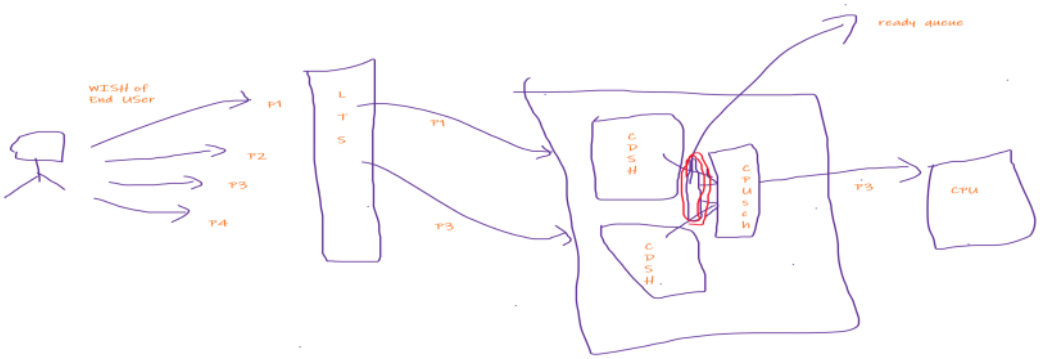
Memory Management --

Memory = primary = RAM = main memory = volatile memory

What is included in the Memory Management ?

1. Keep track of which addresses are allocated and which are free
2. Decide which processes get to run first = High Level Schedular , Long Term Schedular

High Level Schedular	Low Level schedular / CPU schedular
Long Term Schedular (LTS)	Short term schedular (dispatcher)
Selects which process must be started, must be loaded in RAM	Selects process from ready queue , so that it can use the CPU



Memory has addresses ------

Physical address, Logical address , base address , actual address , offset address

Physical Address of RAM	Space
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
…	
…	
8GB	

PHYSICAL ADDRESS = address of the RAM location , it starts from 1 ,

LOGICAL / OFFSET ADDRESS = address internal to each process, it starts from 0

Process 1------ Logical/offset address	
0	Instruction1
1	Instruction2
2	Data1
3	Data2
4	Data3
….	
….	

Process 2------ Logical/offset address	
0	Instruction1
1	Instruction2
2	Instruction3
3	Instruction4
4	Data1
5	Data2
6	Data3

Process 3------ Logical/offset address	
0	Instruction1
1	Instruction2
2	Instruction3
3	Instruction4
4	Data1
5	Data2
6	Data3
7	Data5
8	Data6
9	Data6

BASE ADDRESS ---- LOCATION OF THE FROM WHERE THE PROCESS IS LOADED in the RAM
Every process has a different base different

P1 - BASE ADDRESS IS 1
P2 - BASE ADDRESS IS 6

LET US say that P1 and P2 are loaded in the RAM by the LTS ----------

Physical Address of RAM	Space
1 (base address of P1)	0 I1 (Process 1)
2	1 I2
3	2 D1
4	3 D2
5	4 D3 (actual = base + offset) (1 + 4 = 5)
6 (base address of P2)	0 I1 (process 2)
7	1 I2
8	2 I3
9	3 I4 (actual = base + offset) (6 + 3 = 9 = 9 actual)
10	4 D1
11	5 D2
12	6 D3
13	
14	
15	

16	
17	
18	
19	
20	

LTS will not allow process 3 to load , WHY ? Because process 3 has size 10 and free space is size 8 . Free < Required

ACTUAL ADDRESS of Instruction /address ----
The actual RAM location where the instruction/data is loaded

_____

Scenario 1 ---- VARIABLE PARTITION SCHEME
FIXED PARTITION SCHEME
In both cases it is assumed that the **entire process will be loaded in RAM and the process will get consecutive memory locations.**

RAM	
8 bytes	Process P1
10 bytes	FREE
7 bytes	P3
5 bytes	FREE

A new process P4 wants to start . The size of P4 is 8 bytes
Q1 . Can LTS allow P4 to load ? NO . Why ? **Free < required**
After some time P2 is over . 10 bytes space is released . So it is FREE. Total free size = 15 bytes

15 bytes are not consecutive. 2 FREE HOLES are created ---------- 10 , 5
If P4 has to be loaded , which FREE HOLE is used ---- 10 > required

If P5 size 4 has to be loaded ? Which FREE HOLE 10 ? Or 5?

Process Allocation Algorithms -- any one may be implemented with a given kernel

FIRST FIT	**then 10 will be selected**	**First hole that is > = required size**
BEST FIT	**Then 5 will selected**	**(hole_size - required_size) is minimum**
WORST FIT	**Then 10 will be selected**	**(hole_size - required_size) is maximum**

Process P5 size 4 has to be loaded ? WHERE ?

First fit	6
Best fit	4
Worst fit	9

RAM	
12	**P1**
6	**FREE**
10	**P2**
9	**FREE**
8	**P3**
4	**FREE**

MOST POPULAR	**WORST FIT**

VARIABLE PARTITION SCHEME --- Load entire process on consecutive RAM locations

RAM is partitioned = RAM is divided
Partitions are of UNEQUAL sizes
Number of Partitions keep changing

Example
At Time 0 At Time5 At time 10 At time 13

Ram	4 partitions

At Time 0

Ram	4 partitions
10	P1
8	P2
5	P3
2	Free

At Time5

RAM	3 partitions
10	P1
8	P2
7	Free

At time 10

RAM	4 partitions
5	P4
5	Free
8	P2
7	Free

At time 13

RAM	5 partitions
5	P4
3	P5
2	Free
8	P2
7	Free

EXAMPLE

RAM	
5	**P1**
4	**P2**
6	**Free**
2	**P3**
4	**Free**

Total FREE SPACE = 10
A process P6 wants to load , size of P6 = 7
Free > required
Will LTS load the P6 ? NO . Why ? Entire process must load in consecutive locations , 7 consecutive locations are not available .

PROBLEM = even though free space is more than required space the process is not loaded as the consecutive space is not enough ========== **EXTERNAL FRAGMENTATION**

Solution to External Fragmentation
Theoretical solution ----------**COMPACTION** = Move all the free spaces to ONE SIDE of the RAM

RAM	
5	**P1**
4	**P2**
6	**Free**
2	**P3**
4	**Free**

==========>>>>
COMPACTION

RAM	
5	P1
4	P2
2	P3
6	Free
4	Free

Disadvantage = difficult to implement !!!

FIXED PARTITION SCHEME ------ load entire process on consecutive locations of RAM

RAM is pre divided into FIXED number of partitions = **FRAMES**
All the FRAMES are of **equal** size = for ex 2 kb or 4 kb

At Time 0

RAM	LOGICALLY divided into FRAMES
2	F1
2	F2
2	F3
2	F4
2	F5
2	F6
2	F7
2	F8
2	F9
2	F10

RAM	LOGICALLY divided into FRAMES
2	F1 P1
2	F2 P1
2	F3 P2
2	F4 P2
2	F5 P2
2	F6 P3
2	F7 P3
2	F8 P3
2	F9

At Time 1 P1 = 4 , P2 =5

P1 = 4
4/size of frame = 4/2 = 2 FRAMES

P2 = 5
5/sizeoffame = 5/2 = 2.5 ==> 3 Frames

ALLOCATION is in terms of FRAMES
P3 = 6 , 6/2 = 3 FRAMES

2	F8
2	F9
2	F10

2	F7 P3
2	F8 P3
2	F9
2	F10

ALLOCATION is in terms of FRAMES
P3 = 6 , 6/2 = 3 FRAMES

In CASE of P2 Frame 5 is not used completely
Because size of process is not exact multiple of frame size
The space is wasted in Frame 5, as it is not allocated to any other process
PROBLEM ----- **INTERNAL FRAGMENTATION = space INTERNAL to frame is wasted**

CAN the Fixed Partition Scheme have External Fragmentation Problem ?

RAM	LOGICALLY divided into FRAMES
2	F1 P1
2	F2 P1
2	F3 P2
2	F4 FREE
2	F5 FREE
2	F6 P3
2	F7 P3
2	F8 P3
2	F9 FREE
2	F10 FREE

P4 size 6 wants to load - 6/size of frame = 6/2 = 3 FRAMES

Can LTS load P4 ?
Total Free Frames = 4 , Required Frames = 3
Free > required
Consecutive frames = 2 < required }}} NOT ALLOWED

EXTERNAL FRAGMENTATION

Fixed Partition Scheme will have INTERNAL + EXTERNAL FRAGMENTATION PROBLEM

Variable Partition Scheme will have ONLY EXTERNAL FRAGMENTATION PROBLEM

Can variable partition scheme have internal fragmentation problem ???? NO , because NO FRAMES are present !!!!

What is the solution to INTERNAL FRAGMENTATION ? NO Solution is available .

What is the practical solution to EXTERNAL FRAGMENTATION ? Load entire process on non consecutive RAM locations.

Variable Partition Scheme is adapted for non consecutive process storage USING = SEGMENTATION

Fixed Partition Scheme is adapted for non consecutive process storage USING = PAGING

SEGMENTATION = The process logical address space is divided into SEGMENTS on the basis of contents !!!!!

Code segment
Data segment
Stack segment
Extra segment

Process-logical address	Segment number	Segment offset	
Code Segment	0	0	I1
		1	I2
		2	I3
		3	I4
Data Segment	1	0	GD1
		1	GD2
Stack Segment	2	0	LD1
		1	LD2

		2	LD3
Extra segment	3	0	DD1
		1	DD2
		2	DD3

Process is divided into UNEQUAL size segments , based on the content
Logical address of I2 = **seg-num** + **seg offset**

RAM		
4	1---5	P1 S1
3	5---8	FREE
5	8---13	P2 S0
7	13----20	P1 S0
8	20---28	P2 S1
4	28---32	P1 S2

HOW will the Kernel Know which segments are WHERE ???

A data structure called as Segment Table is created FOR EACH Process !!!!!!!!

P1 Segment TABLE

seg-number	Segment -size	Segment-base-address
0	7	13
1	4	1
2	4	28

P2 Segment Table

Seg-number	Segment-size	Seg-base-address
0	5	8
1	8	20

ACTUAL ADDRESS OF INSTRUCTION P1
PC = register in CPU -- holds the address of next instruction
SEGMENT NUMBER = 0
SEGMENT OFFSET = 3

ACTUAL ADDRESS = use seg number and seg table to find seg base
= seg base + seg offset
= 13 + 3 = **16** } the instruction to be loaded is on actual physical location 16

ADVANTAGE -------

1. NO Internal fragmentation , NO FRAMES
2. External Fragmentation is REDUCED to a great extent
   Sometimes the **segment size < consecutive free size** , so again we face external fragmentation
3. Segment table size is small , few entries

Disadvantage ----------

1. External fragmentation is still present , not totally overcome
2. Overhead of segment table maintenance

PAGING ----------entire process is loaded in RAM but in **non-consecutive** locations in RAM

As this is based on fixed partition scheme ---- FRAMES

RAM is divided into equal size fix number of frames!!!

RAM	LOGICALLY divided into FRAMES
2	F1
2	F2
2	F3
2	F4
2	F5

2	F6
2	F7
2	F8
2	F9
2	F10

Process is divided into PAGES based on **size** (same as frame size) !!!

Process Logical addresses	Page Number	Page Offset	
	0	0	I1
		1	I2
	1	0	I3
		1	I4
	2	0	I5
		1	D1
	3	0	D2
		1	D3
	4	0	D4
		1	D5
	5	0	D6
		1	D7

RAM

F1	P1 Pg0
F2	P2Pg3
F3	P1 Pg1
F4	Free
F5	P1 pg2
F6	P1 Pg3
F7	P2 Pg0
F8	P1Pg4
F9	P2 Pg2
F10	P1 Pg5

How will the Kernel keep track of all the pages of the process ?

Maintain a data structure PAGE TABLE for EACH process

P1 Page Table

Page-number	Frame-number
0	1
1	3
2	5
3	6
4	8
5	10

P2 Page Table

Page-Number	Frame-number
0	7
1	
2	9
3	2

Paging

Advantages

1. If non consecutive Free frames > required frames process can be loaded ---NO EXTERNAL FRAGMENTATION , totally REMOVED

Disadvantages

1. Internal Fragmentation is present
2. **Size of page table is very large . Page Table management techniques have to be used**

Book

Chapter ----division ---------- content wise ---- SEGMENTATION 10 chapter --- segment table is small

Pages ----- division ---- size wise ---- PAGING ---------- 500 pages --- page table is large

---