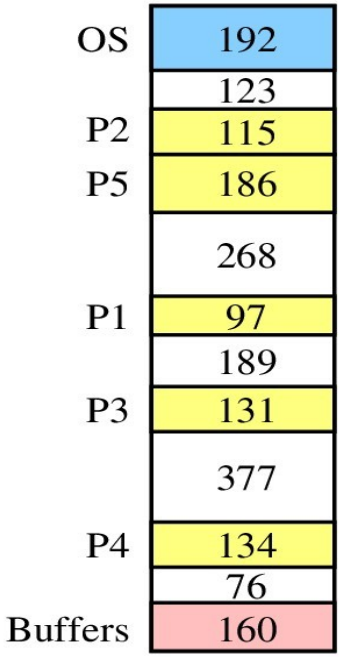
1. **[Contiguous allocation]**

New processes:

P6 (180 MB)
P7 (70 MB)
P8 (210 MB)

Show allocations for:

(a) First fit
(b) Best fit
(c) Worst fit



See EndSem 2024 Solution

2. Suppose that a system supports 1 KB pages. The logical address has a size of 40 bits. Each entry in the page table takes 4 bytes.

(a) What is the size of the logical address space of a process?

2^{40} bytes = 1 TB.

(b) How many entries are needed in the page table of a process? What is the size of each page table?

2^{40} bytes / 1 KB = 2^{30}.
$2^{30} \times 4$ bytes = 4 GB.

(c) Now, suppose that hierarchical paging is used. How many levels in the hierarchy do you need?

Each page can store 1 KB / 4 byte = 256 = 2^8 entries of the page table. We have 30 = 6 + 8 + 8 + 8, so a four-level hierarchy is needed.

(d) If we use an inverted page table with each entry having 8 bytes, and we have 8 GB physical memory, what will be the size of the inverted page table?

No of entries in the inverted page table = 8 GB / 1 KB = $8 \times 2^{20} = 2^{23}$.
Size of inverted page table = $2^{23} \times 8$ bytes = 2^{26} bytes = 64 MB.

3. [IA-32 Segmentation]

A segment descriptor (local or global) consists of 64 bits. It contains a 32-bit base (a 32-bit address in the linear address space), and a 20-bit limit (size of the segment). One of the remaining 12 bits is called the granularity bit *G* which helps us interpret the 20-bit limit. If $G = 0$, then the granularity is byte, so the size of the segment is restricted to 2^{20} bytes, that is, 1 MB. If $G = 1$, then the granularity is 4 KB (page size for IA-32), so the maximum size of a segment is 4 GB. The base address can be any 32-bit value. However, it is recommended to be 16-byte aligned. Moreover, if we are dealing with $G = 1$ for all segments, the base address should be page aligned.

Now, suppose that a process uses the following segments. Assume that each is a local segment, and so the descriptors are in the local descriptor table.

Segment No	Segment size (bytes)
0	0x1234
1	0xbcdef
2	0x567
3	0x2b3c4
4	0x7531

(a) Show the relevant portions of the LDT in the following two cases.

Case 1: $G = 0$ with 16-byte alignment of all the segments
Case 2: $G = 1$ with 4 KB alignment of all the segments

In both the cases, assume that the segments are located one after another starting from linear address 0. In order to maintain the alignment, we only use a minimal amount of internal fragmentation.

Segment sizes (in bytes)

Segment No	Actual size	Rounded up size (Case 1)	Rounded up size (Case 2)
0	0x1234	0x1240	0x2000
1	0xbcdef	0xbcdf0	0xbd000
2	0x567	0x570	0x1000
3	0x2b3c4	0x2b3d0	0x2c000
4	0x7531	0x7540	0x8000

Local Descriptor table

Case 1

Segment No	*G*	Base	Limit
0	0	0x00000000	0x01240
1	0	0x00000000 + 0x1240 = 0x00001240	0xbcdf0
2	0	0x00001240 + 0xbcdf0 = 0x000be030	0x00570
3	0	0x00be030 + 0x570 = 0x000be5a0	0x2b3d0
4	0	0x00be5a0 + 0x2b3d0 = 0x000e9970	0x07540

Case 2

Segment No	*G*	Base	Limit
0	1	0x00000000	0x00002
1	1	0x00000000 + 0x2000 = 0x00002000	0x000bd
2	1	0x00002000 + 0xbd000 = 0x000bf000	0x00001
3	1	0x000bf000 + 0x1000 = 0x000c0000	0x0002c
4	1	0x000c0000 + 0x2c000 = 0x000ec000	0x00008

(b) Assume that in each case of Part (a), the linear address space is paged. Show the page-level organization of the linear address space. Take a 4 KB page size.

Case 1

Segment 0	Full of page 0, first 240 – 1 = 23f bytes in Page 1
Segment 1	Next 1000 – 240 = dc0 bytes in Page 1, all of pages 2, 3, . . . , bd, first 30 bytes of Page be
Segment 2	Bytes 30 to 30 + 570 – 1 = 59f of Page be
Segment 3	Bytes 5a0 onward of Page be, all of pages bf through e8, first 970 bytes of Page e9
Segment 4	Bytes 970 onward in Page e9, all of pages ea through ef, first eb0 bytes if Page f0

Case 2

Segment 0	Pages 0 and 1
Segment 1	Pages 2 through 2 + bd – 1 = be
Segment 2	Page bf
Segment 3	Pages c0 through eb
Segment 4	Page ec through f3

(c) For each of the two cases, convert the following logical address to linear addresses.

(0, 0x234)
(1, 0x11111)
(2, 0x100)
(3, 0x2b004)
(4, 0x7539)
(4, 0x7549)

Case 1

(0, 0x234) → 0x00000000 + 0x234 = 0x00000 234
(1, 0x11111) → 0x00001240 + 0x11111 = 0x00012 351
(2, 0x100) → 0x000be030 + 100 = 0x000be 130
(3, 0x2b004) → 0x00be5a0 + 0x2b004 = 0x000e9 594
(4, 0x7539) → 0x000e9970 + 0x7539 = 0x000f0 ea9
(4, 0x7549) → Invalid address

Case 2

(0, 0x234) → 0x00000000 + 0x234 = 0x00000 234
(1, 0x11111) → 0x00002000 + 0x11111 = 0x00013 111
(2, 0x100) → 0x000bf000 + 0x100 = 0x000bf 100
(3, 0x2b004) → 0x000c0000 + 0x2b004 = 0x000eb 004
(4, 0x7539) → 0x000ec000 + 0x7539 = 0x000f3 539
(4, 0x7549) → 0x000ec000 + 0x7549 = 0x000f3 549