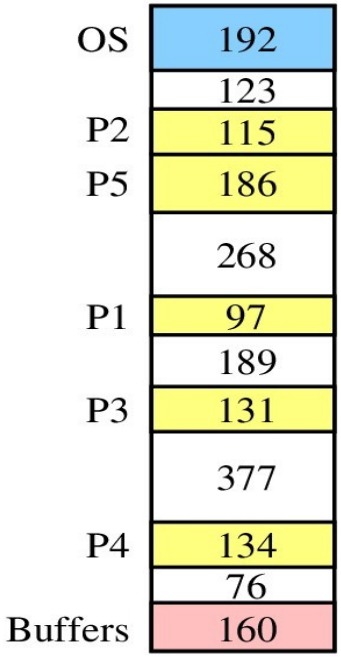
1. [*Contiguous allocation*]

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. [*Contiguous allocation*]

A system implements the contiguous memory allocation scheme. At some point of time, there are only two available holes of sizes 300MB and 500MB. Then, three processes P1, P2, and P3 arrive (in that order) with memory requirements x MB, y MB, and z MB, respectively. Supply explicit integer values to x, y, and z in order to demonstrate each of the following two situations. In each case, also explain the details how hole allocations proceed or fail. No credit for trying to prove that some values of x, y, z (may) exist. Assume that when a hole is given to a process which does not fill the entire hole, only one new hole is created (at one end of the old hole).

Situation 1: The best-fit strategy can give the required memory to all of the three processes, whereas the worst-fit strategy can give the required memory only to P1 and P2 (P3 has to wait).

$x = 200,\ y = 100,\ z = 450$

Situation 2: The worst-fit strategy can give the required memory to all of the three processes, whereas the best-fit strategy can give the required memory only to P1 and P2 (P3 has to wait).

$x = 200,\ y = 300,\ z = 250$

3. 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.

4. [*IA-32 Segmentation*]

Consider the IA32 segmentation scheme along with paging of the linear address space into 4KB pages. Suppose that a process has one thousand 1KB segments numbered 0, 1, 2, . . . , 999, one hundred 10KB segments numbered 1000, 1001, 1002, . . . , 1099, ten 100KB segments numbered 1100, 1101, 1102, . . . , 1109, and one 1MB segment numbered 1110. Assume that all these segments are local to the process, and that the process uses no global segments. In the linear address space, the segments are placed in the sequence of the segment numbers, starting from linear address 0, and are aligned at page boundaries. Suppose that all these pages are loaded to memory frames (no demand paging). In this context, answer the following questions. Show your calculations.

(a) How much space is lost due to internal fragmentation?

Each of the 1KB segments is mapped to a page/frame of size 4KB, leading to a total internal fragmentation of 3000KB for these segments.

Each of the 10KB segments needs three 4KB pages/frames, so the total internal fragmentation for these segments is 200KB.

Since 100KB and 1MB are exact multiples of 4KB, no internal fragmentation results from segments of these sizes.

To sum up, total internal fragmentation is 3000KB + 200KB = 3200KB.

(b) What is the total size of the linear address space (including internal fragmentation)?

$1000 \times 4\text{KB} + 100 \times 12\text{KB} + 10 \times 100\text{KB} + 1\text{MB} = 7224\text{KB}$.

(c) The linear address space is paged. What will be the size of the page table (that is, the number of entries in the page table) of the process?

7224KB / 4KB = 1806.

(d) IA32 uses two-level hierarchical paging using the 10 + 10 + 12 breakup of the form page-directory (outer-page) index + page-table (inner-page) index + offset. Also suppose that each page-table entry is of size 32 bits. Explain how the linear address space of the process is paged. Clearly calculate the numbers of entries needed in the page directory and in all the (inner) page tables.

One frame can store 4KB / 4B = 1024 entries. Two entries are needed in the page directory (outer page table) for 1806 pages. The first entry will point to a page table with 1024 frame pointers, and the second entry will point to a page table with 1806 – 1024 = 782 frame pointers.

(e) Explain how the logical address (1024, 4201) [this is in the format (segment number, offset)] is converted to a linear address and then to a physical address.

The segment 1024 is a 10KB segment. Before it, appear 1000 1KB segments (each having one page) and 24 10KB segments (each having three pages). So this segment starts from page 1000 + 3 × 24 = 1072 of the linear address space. The offset is 4201 = 4096 + 105. So the linear address is (1073, 105). As a single 32-bit value, the linear address is 1073 × 4096 + 105 = 4395113.

In order to map this linear address to a physical address, we look at the page directory. Since 1073 > 1024, we follow the second entry of the page directory. From this page table, we look at the entry 1073 – 1024 = 49 (numbering starts from 0). The page-table entry gives the frame number of the address we started with. The offset in that frame will be 105.

5. [*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