Answers of Chapter (8)

8.1 Explain the difference between internal and external fragmentation.

Answer: Internal Fragmentation is the area in a region or a page that is not used by the job ***occupying*** that region or page. This space is unavailable for use by the system until that job is finished and the page or region is released. The main difference is the ***allocation***: internal fragmentation is allocated area and unused but external fragmentation is un allocated area and unused.

* * * *

8.16 Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)?Which algorithm makes the most efficient use of memory?

Answer:

Let p1, p2, p3 & p4 are the names of the processes

a. First-fit:

- **P1>>>** 100, **500**, 200, 300, 600
- **P2>>>** 100, 288, 200, 300, **600**
- **P3>>>** 100, **288**, 200, 300, 183
- 100, 116, 200, 300, 183 **<<<<< final set of hole**
- P4 (426K) must wait

b. Best-fit:

- **P1>>>** 100, 500, 200, **300**, 600
- **P2>>>** 100, **500**, 200, 88, 600
- **P3>>>** 100, 83, **200**, 88, 600
- **P4>>>** 100, 83, 88, 88, **600**
- 100, 83, 88, 88, 174 **<<<<< final set of hole**

c. Worst-fit:

- **P1>>>** 100, 500, 200, 300, **600**
- **P2>>>** 100, **500**, 200, 300, 388
- **P3>>>** 100, 83, 200, 300, **388**
- 100, 83, 200, 300, 276 **<<<<< final set of hole**
- P4 (426K) must wait

In this example, Best-fit turns out to be the best ***because there is no wait processes.***

* * * *

٨,١٤ Consider a logical address space of 64 pages of 1024 words each، mapped onto a physical memory of 32 frames.
a. How many bits are there in the logical address؟
b. How many bits are there in the physical address؟

Answer:

Method1:

a) m =???

Size of logical address space $= 2^m$ = # of pages × page size

$2^m = 64 \times 1024$

$2^m = 2^6 \times 2^{10}$

$2^m = 2^{16}$ »» **m=16 bit**

Method2:

m =???

# of pages = 2^{m-n}

n =???

Page size=2^n

$1024=2^n$

$2^{10}=2^n$ »» n=10 bit

Again: # of pages = 2^{m-n}

$64=2^{m-10}$

$2^6=2^{m-10}$

6= m-10 »» **m=16 bit**

b)

Let **(x)** is number of bits in the physical address

x =???

Size of physical address space = 2^x

Size of physical address space = # of frames × frame size

(frame size = page size)

Size of physical address space = 32 × 1024

$2^x=2^5 \times 2^{10}$

$2^x = 2^{15}$

»» number of required bits in the physical address=**x =15 bit**

* * * *

8.22 Consider a logical address space of 32 pages of 1024 words per page, mapped onto a physical memory of 16 frames.
a. How many bits are required in the logical address?
b. How many bits are required in the physical address?

Answer:

a) m =???

Size of logical address space $= 2^m$ = # of pages × page size

= 32 × 1024 = 2^{15} »» m=15 bit

b) Size of physical address space = # of frames × frame size
(frame size = page size)

Size of physical address space = $16 \times 1024 = 2^{14}$
»» number of required bits in the physical address =14 bit

* * * *

8.19 Assuming a 1-KB page size , What are the *page numbers* and *offsets* for the following address references (provided as decimal numbers)

a. 2375
b. ١٩٣٦٦
c. 30000
d. 256
e. 16385

Answer:
Page size $=2^n = 1024\ B = 2^{10}\ B$
# of bits in offset part (n) =10

Solution steps :

1. Convert logical address: Decimal → Binary
2. Split binary address to 2 parts (page # , Offset), offset : *n* digits
3. Convert offset & page# : Binary→ Decimal

Logical address (decimal)	*Logical address (binary)*	*Page # (6 bits) (binary)*	*Offset (10 bits) (binary)*	*Page # (decimal)*	*Offset (decimal)*
2375	0000 1001 0100 0111	0000 10	01 0100 0111	2	327
19366	0100 1011 1010 0110	0100 10	11 1010 0110	18	934
30000	0111 0101 0011 0000	0111 01	01 0011 0000	29	304
256	0000 0001 0000 0000	0000 00	01 0000 0000	0	256
16385	0100 0000 0000 0001	0100 00	00 0000 0001	16	1

* * * *

8.10 Consider a paging system with the page table stored in memory.
a. If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?
b. If we add associative registers, and 75 percent of all page-table references are found in the associative registers, what is the effective memory reference time? (Assume that finding a page-table entry in the associative registers takes zero time, if the entry is there.)

Answer:
a. memory reference time= 200+200= 400 ns
(200 ns to access the page table in RAM and 200 ns to access the word in memory)

b.
Case (1) : page entry found in associative registers (part1)
Memory access time = 0+200=200 ns
(0 ns to access the page table in associative registers and 200 ns to access the word in memory)

Case (2) : page entry NOT found in associative registers (part1) but found in page table in RAM
Memory access time = 0+200+200=200 ns
(0 ns to access the page table in associative registers (part1) ,200 ns to access the page table(part2) in RAM and 200 ns to access the word in memory)

>>> Effective access time =∑ [probability of the case × access time of this case]
Effective access time = [0.75 × 200]+ [0.25 × 400]= 250 ns.

* * * *

8.8 On a system with paging, a process cannot access memory that it does not own; why? How could the operating system allow access to other memory? Why should it or should it not?

Answer:
- On a system with paging, a process cannot access memory that it does not own; why?
An address on a paging system is a logical page number and an offset. The physical page is found by searching a table based on the logical page number to produce a physical page number. ***Because the operating system controls the contents of this table, it can limit a process to accessing only those physical pages allocated to the process.*** There is no way for a process to refer to a page it does not own because the page will not be in the page table.

-How could the operating system allow access to other memory?
To allow such access, an operating system simply needs to ***allow entries for non-process memory to be added to the process's page table.***

- Why should it or should it not?
This is useful ***when two or more processes need to exchange data***—they just read and write to the same physical addresses (which may be at varying logical addresses). This makes for very efficient interprocess communication.
It's useful also to implement ***sharing.***

* * * *

8.15 Consider the hierarchical paging scheme used by the VAX architecture. How many memory operations are performed when an user program executes a memory load operation?

Answer: VAX is ***2-level hierarchical paging***
When a memory load operation is performed, there are *three memory operations* that might be performed:

1. One is access the *outer page table* .
2. The second access is to access the page table entry itself *(inner page table)*
3. The third access is the actual memory load operation.

* * * *

8.6 What is the purpose of paging the page tables?

Answer: In certain situations the ***page tables could become large*** enough that by paging the page tables, one could simplify the memory allocation problem and also enable the ***swapping of portions of page table that are not currently used***.

Note: see the example in book in the beginning of sec.# (8.5.1) (system with 32-bit logical address…….etc)

* * * *

8.18 Consider a computer system with a 32-bit logical address and 4-KB page size . The system supports up to 512MB of physical memory. How many entries are there in each of the following:
a. A conventional single-level page table
b. An inverted page table

Answer:
a) # of pages= # of entries =????
Size of logical address space $= 2^m$ = # of pages × page size
»» 2^{32} = # of pages × 2^{12}
# of pages =2^{32} / 2^{12}= 2^{20} pages

b)
Size of physical address space = # of frames × frame size
(frame size = page size) »» »» »» In paging in general
(# of frames = # of pages) »» »» »» In Inverted paging ONLY

2^{29}= # of frames × 2^{12}
# of frames =2^{29} / 2^{12}= 2^{17} pages
Number of entries= = 2^{17} pages

* * * * *

8.21 Consider the following segment table:

Segment	Base	Length	
0	219	600	
1	2300	14	
2	90	100	>> there is a typing error in this row in book

What are the physical addresses for the following logical addresses?
a. 0,430
b. 1,10
c. 2,500

Answer:
a. 219 + 430 = 649
b. 2300 + 10 = 2310
c. illegal reference, trap to operating system

* * * *

8.7 Explain why sharing a reentrant module is easier when segmentation is used than when pure paging is used

Answer: Since segmentation is based on a logical division of memory rather than a physical one, segments of any size can be shared ***with only one entry in the segment tables of each process*** . With paging there must be a common entry in the page tables for each page that is shared (i.e. operating system needs to add ***multiple page entries*** in the page table to share the module)

* * * *

8.17 Describe a mechanism by which one segment could belong to the address space of two different processes.

Answer: Since segment tables are a collection of base–limit registers, segments can be shared when entries in the segment table of two different jobs point to the same physical location. *The two segment tables must have identical base and limit values, and the shared segment number must be the same in the two processes.*

* * * *

8.11 Compare paging with segmentation with respect to the amount of memory required by the address translation structures in order to convert virtual addresses to physical addresses.

Answer: *Paging requires more memory* overhead to maintain the translation structures. Segmentation requires just one (or more) entry for code and one entry for data. Paging on the other hand requires **multiple entries for code and data** depending on page size. So , paging almost requires more memory for page table than the memory space required to segment table.

* * * *

8.2 Compare the main memory organization schemes of contiguous-memory allocation, pure segmentation, and pure paging with respect to the following issues:
a. external fragmentation
b. internal fragmentation
c. ability to share code across processes

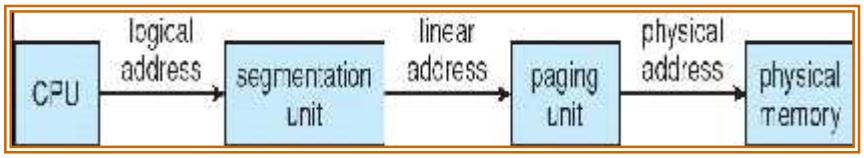
Answer:

Method	External fragmentation	Internal fragmentation	ability to share code
contiguous-memory allocation (*variable size method*)	There is external fragmentation (as address spaces are allocated contiguously and holes develop as finished processes release its space and new processes are allocated and the size of the new process is almost smaller than the old one)	There is no internal fragmentation	It does not allow processes to share code.
pure paging	There is no external fragmentation	There is internal fragmentation (it appears in the last frame because the process size almost not a multiplex of page size)	Able to share code between processes
pure segmentation	There is external fragmentation (fragmentation would occur as segments of finished processes are replaced by segments of new processes. and the size of the new process is almost smaller than the old one)	There is no internal fragmentation	Able to share code between processes

* * * * *

8.5 Consider the Intel address translation scheme shown in Figure 8.22.
a. Describe all the steps that the Intel 80386 takes in translating a logical address into a physical address.
b. What are the advantages to the operating system of hardware that provides such complicated memory translation hardware?
c. Are there any disadvantages to this address-translation system? If so, what are they? If not, why is it not used by every manufacturer?

Answer:



a. ***stage#1 (segmentation):*** The selector is an index into the segment descriptor table. The segment descriptor result plus the original offset is used to produce a linear address .
stage#2 (2-level paging): The linear address is divided to 3 parts: dir, page, and offset. The dir is an index into a page directory (outer page table). The entry from the page directory selects the page table (inner page table). The entry from the page table, plus the offset, is the physical address.

b. If this complicated memory translation can be done in hardware, it is more efficient (***less time and overhead*** in operating system).

c. Address translation can take ***longer time*** due to the multiple table lookups it can invoke.

* * * * *

>>How to convert from logical address to physical address in paging???

Physical address = base address+ offset
= (frame # * frame size) + offset

IMPORTANT NOTE:

Paging arithmetic laws:

Page size = frame size

Logical address space (/size) = 2^m

Physical address space (/size) = 2^x (*where* ***x*** *is the number of bits in physical address*)

Logical address space (/size) = # of pages × page size

Physical address space (/size) = # of frames × frame size

Page size= frame size= 2^n

# of pages= 2^{m-n}

# of entries (records)in page table = # of pages

(# of frames = # of pages) »» »» »» In Inverted paging ONLY

>>How to convert from logical address to physical address in segmentation???

IF (offset ≤ limit) then
Physical address= base address+ offset
Else
Trap (address error)

Effective access time =
∑ [probability of the case × access time of this case]

Where (∑ probability of all cases =100%)