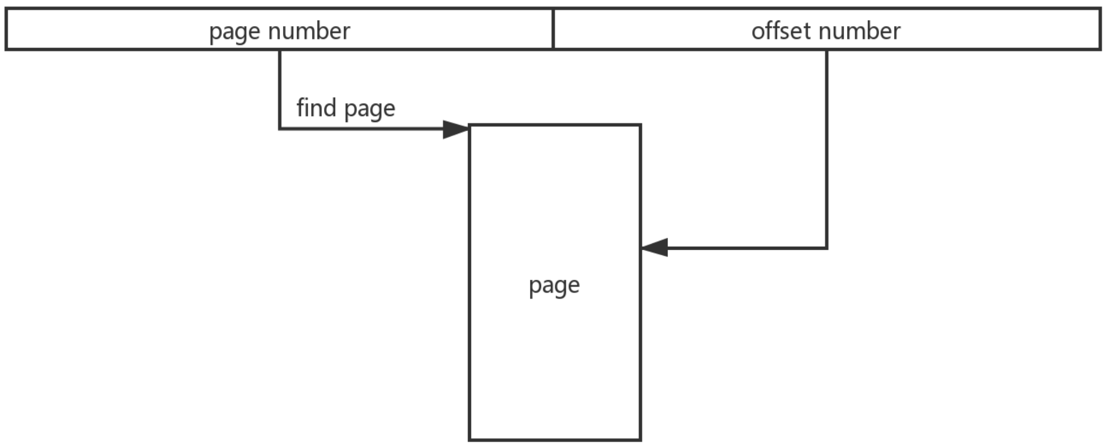
# HW4

## *Chapter 9*

### 9.2

**Question:** Why are page sizes always powers of 2?

**Answer:** Firstly, we have to figure out that paging is implemented by breaking up an address into a page and offset number. Obviously, the most efficient way to get page and offset number from the address is split like the illustration shown rather than do any other additional arithmetic operation. And computers use binary which means that each bit position represents a power of 2. So the page sizes always powers of 2.



### 9.4

**Question:** Consider a logical address space of 64 pages of 1,024 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:** In order to answer this question clearly, we need to do some calculation.(Assume the address corresponds to a word, if the address corresponds to a byte, 2 more bits will be required)

Logical address: 16 bits

$$\because\ 64\,pages = 2^6\ pages$$
$$\therefore\ we\ need\ 6\ bits\ to\ specify\ the\ page\ number$$
$$\because\ 1024\,words = 2^{10}\ words$$
$$\therefore\ we\ need\ 10\ bits\ to\ specify\ the\ word\ number(offset)$$
$$\therefore\ 6+10=16\ bits$$

Physical address: 15 bits

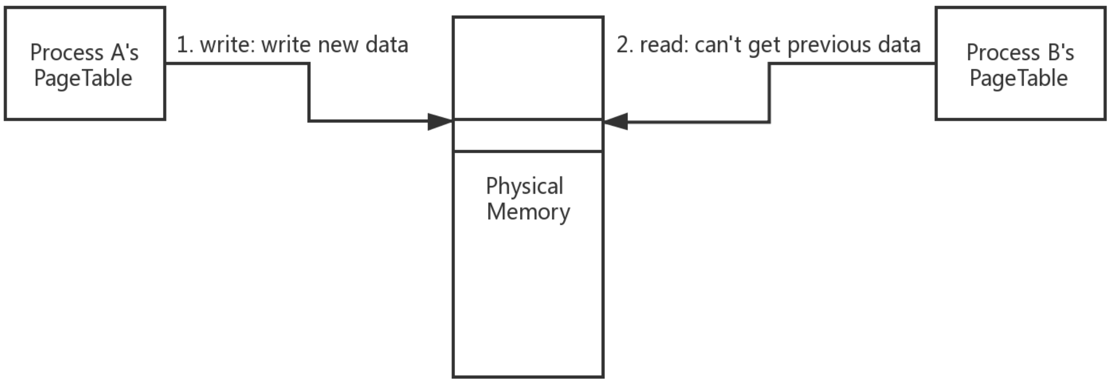
$$\because\ 32\ frames = 2^5\ frames$$
$$\therefore\ we\ need\ 5\ bits\ to\ specify\ the\ frame\ number$$
$$\because\ 1024\,words = 2^{10}\ words\ (Page\ size = Frame\ size)$$
$$\therefore\ we\ need\ 10\ bits\ to\ specify\ the\ word\ number(offset)$$
$$\therefore\ 5+10=15\ bits$$

## 9.5

**Question:** What is the effect of allowing two entries in a page table to point to the same page frame in memory? Explain how this effect could be used to decrease the amount of time needed to copy a large amount of memory from one place to another. What effect would updating some byte on one page have on the other page?

**Answer:** The effect of allowing two entries in a page table to point to the same page frame in memory is that we can avoid many unnecessary memory space cost on save the same code and data, especially for those large programs which have many reentrant code. In order to utilize this effect, when we copy large amounts of memory we can use different page tables point to the same memory location. However, with this approach we can't promise that the correction of the data and code, as any user(process) which hold the page tables point to those same memory location can modify the content which located there.

One case is shown in the illustration below.



## 9.6

**Question:** Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)?

**Answer:**

**a. first-fit:** allocate the first hole that is big enough

| Process | Memory | | | | | |
|---|---|---|---|---|---|---|
| P1 = 115 KB | M1 = 300 KB | M2 = 600 KB | M3 = 350 KB | M4 = 200 KB | M5 = 750 KB | M6 = 125 KB |
| P2 = 500 KB | M1 = 185 KB | M2 = 600 KB | M3 = 350 KB | M4 = 200 KB | M5 = 750 KB | M6 = 125 KB |
| P3 = 358 KB | M1 = 185 KB | M2 = 100 KB | M3 = 350 KB | M4 = 200 KB | M5 = 750 KB | M6 = 125 KB |
| P4 = 200 KB | M1 = 185 KB | M2 = 100 KB | M3 = 350 KB | M4 = 200 KB | M5 = 392 KB | M6 = 125 KB |
| P5 = 375 KB | M1 = 185 KB | M2 = 100 KB | M3 = 150 KB | M4 = 200 KB | M5 = 392 KB | M6 = 125 KB |

**b. best-fit:** allocate the smallest hole that is big enough

| Process | Memory | | | | | |
|---|---|---|---|---|---|---|
| P1 = 115 KB | M1 = 300 KB | M2 = 600 KB | M3 = 350 KB | M4 = 200 KB | M5 = 750 KB | M6 = 125 KB |
| P2 = 500 KB | M1 = 300 KB | M2 = 600 KB | M3 = 350 KB | M4 = 200 KB | M5 = 750 KB | M6 = 10 KB |
| P3 = 358 KB | M1 = 300 KB | M2 = 100 KB | M3 = 350 KB | M4 = 200 KB | M5 = 750 KB | M6 = 10 KB |
| P4 = 200 KB | M1 = 300 KB | M2 = 100 KB | M3 = 350 KB | M4 = 200 KB | M5 = 392 KB | M6 = 10 KB |
| P5 = 375 KB | M1 = 300 KB | M2 = 100 KB | M3 = 150 KB | M4 = 200 KB | M5 = 392 KB | M6 = 10 KB |

**c. worst-fit:** allocate the largest hole

| Process | Memory | | | | | |
|---|---|---|---|---|---|---|
| P1 = 115 KB | M1 = 300 KB | M2 = 600 KB | M3 = 350 KB | M4 = 200 KB | M5 = 750 KB | M6 = 125 KB |
| P2 = 500 KB | M1 = 300 KB | M2 = 600 KB | M3 = 350 KB | M4 = 200 KB | M5 = 635 KB | M6 = 125 KB |
| P3 = 358 KB | M1 = 300 KB | M2 = 600 KB | M3 = 350 KB | M4 = 200 KB | M5 = 135 KB | M6 = 125 KB |
| P4 = 200 KB | M1 = 300 KB | M2 = 242 KB | M3 = 350 KB | M4 = 200 KB | M5 = 135 KB | M6 = 125 KB |
| P5 = 375 KB | M1 = 300 KB | M2 = 42 KB | M3 = 150 KB | M4 = 200 KB | M5 = 135 KB | M6 = 125 KB |

As the table shown, using worst-fit algorithm will cause that Process 5 whose size is 375 KB must wait.

## 9.15

**Question:** Compare the memory organization schemes of contiguous memory allocation and paging with respect to the following issues:
  a. External fragmentation
  b. Internal fragmentation
  c. Ability to share code across processes

**Answer:**

**contiguous memory allocation**

If the address space is allocated with the contiguous memory allocation scheme, as old processes die and new processes are initiated, in the original contiguous allocated space some holes will appear, which can easily become or bring **external fragmentation**(but no internal fragmentation). And it will make the code sharing across processes impossible, because the virtual space in this situation is an  indivisible body, we can't pick up the code part.

**paging without segmentation**

If he address space is allocated with the paging scheme without segmentation. Although it solve the external fragmentation but **internal fragmentation** will become a severe problem, and this kind of page maybe need to be set in a big size if a process need a large space. Because the same reason of contiguous memory allocation, this scheme can't realize **code sharing across processes**  as well.

**paging with segmentation**

With this scheme, we can make the page size be more appropriate, which can well improve the **internal fragmentation** problem. As we can make the code in a independent segment, **code sharing across processes** can be realized now.

## 9.19

**Question:** Explain why address-space identifiers (ASIDs) are used in TLBs.

**Answer:** Because we use address-space identifiers (ASIDs)  as the only sign of a process in TLBs. When TLB try to analyze the virtual page number, it will ensure the the current process's ASID is same with the ASID relative to the page, or the TLB will be set to invalid. This mechanism **provide address space protection** in the TLB.

What's more, as we use address-space identifiers (ASIDs) to distinguish entries belong to which process, TLB can **hold several different processes' entries at the same time**.

## 9.20

**Question:** Program binaries in many systems are typically structured as follows. Code is stored starting with a small, fixed virtual address, such as 0. The code segment is followed by the data segment, which is used for storing the program variables. When the program starts executing, the stack is allocated at the other end of the virtual address space and is allowed to grow toward lower virtual addresses. What is the significance of this structure for the following schemes?
  a. Contiguous memory allocation
  b. Paging

**Answer:**

a. **Contiguous memory allocation**

If we use contiguous-memory allocation scheme, and code is stored starting with a small, the stack is allocated at the other end, it will lead to that the operating system allocate the entire virtual address space to the program when it starts to be executed. And we can easily find that in the most situation the whole virtual address space is much larger than the number of a process's address space requirement. It will cause severe space waste.

b. **Paging**

If we use paging scheme instead of contiguous-memory allocation. One improvement is that the operating system do not need to allocate the maximum extent of the virtual address space to a single process at startup time, but if paging work without segmentation, it still requires the operating system to allocate a large page table which can cover all of the program’s virtual address space. When he stack or the heap of a program needs to grow toward lower virtual addresses, a problem will occur, the corresponding page table entry is preallocated so we can't allocate a new page anymore. But if we can combine segmentation with paging, these problems can be solved, and the allocation can be more flexible.

## 9.25

**Question:** Consider a paging system with the page table stored in memory.
a. If a memory reference takes 50 nanoseconds, how long does a paged memory reference take?
b. If we add TLBs, and if 75 percent of all page-table references are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes 2 nanoseconds, if the entry is present.)

**Answer:**

a. 100 nanoseconds.  Because we need to do memory reference operation two times one for accessing the page table and the other for accessing the word in memory.

$$50ns + 50ns = 100ns$$

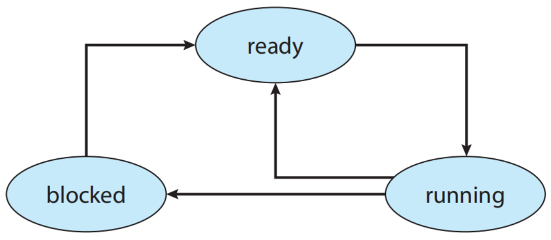
b. 64 nanoseconds.  75% of the time it's (2+50) nanoseconds, and the other 25% of the time it's 100 nanoseconds. And the calculation process is shown below:

$$\begin{aligned} effective\ memory\ reference\ time\ &= 75\% * (2ns + 50ns) + 25\% * (2ns + 50ns + 50ns) \\ &= 64.5ns \end{aligned}$$

# Chapter 10

## 10.16

**Question:** A simplified view of thread states is ready,running, and blocked, where a thread is either ready and waiting to be scheduled, is running on the processor, or is blocked (for example, waiting for I/O).



Assuming a thread is in the running state, answer the following questions, and explain your answers:
a. Will the thread change state if it incurs a page fault? If so, to what state will it change?
b. Will the thread change state if it generates a TLB miss that is resolved in the page table? If so, to what state will it change?
c. Will the thread change state if an address reference is resolved in the page table? If so, to what state will it change?

**Answer:**

a. Yes, if a page fault happens, the thread will change state. When a page fault comes up in the the running state , the thread state will be set to blocked because in this situation an I/O operation is required to bring the new page which is not in the main memory and is necessary for the thread running into memory.

b. No, the state will not change. Because a TLB miss that is resolved in the page table will not cause page fault, the effect is just that memory reference time will be longer. The thread will continue running.

c. No, the state will not change if an address reference is resolved in the page table. It is a normal situation. The thread will continue running.

## 10.24

**Question:** Apply the (1) FIFO, (2) LRU, and (3) optimal (OPT) replacement algorithms for the following page-reference strings:
• 2, 6, 9, 2, 4, 2, 1, 7, 3, 0, 5, 2, 1, 2, 9, 5, 7, 3, 8, 5
• 0, 6, 3, 0, 2, 6, 3, 5, 2, 4, 1, 3, 0, 6, 1, 4, 2, 3, 5, 7
• 3, 1, 4, 2, 5, 4, 1, 3, 5, 2, 0, 1, 1, 0, 2, 3, 4, 5, 0, 1
• 4, 2, 1, 7, 9, 8, 3, 5, 2, 6, 8, 1, 0, 7, 2, 4, 1, 3, 5, 8
• 0, 1, 2, 3, 4, 4, 3, 2, 1, 0, 0, 1, 2, 3, 4, 4, 3, 2, 1, 0
Indicate the number of page faults for each algorithm assuming demand paging with three frames.

**Answer:**

**FIFO:**

- 2; 2,6; 2,6,9; 6,9,4; 9,4,2; 4,2,1; 2,1,7; 1,7,3; 7,3,0; 3,0,5; 0,5,2; 5,2,1; 2,1,9; 1,9,5; 9,5,7; 5,7,3; 7,3,8; 3,8,5(**18 page faults**)

| | **2** | **6** | **9** | **2** | **4** | **2** | **1** | **7** | **3** | **0** | **5** | **2** | **1** | **2** | **9** | **5** | **7** | **3** | **8** | **5** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 5 | 9 | 9 | 9 | 3 | 3 | 3 |
| frame2 | | 6 | 6 | 6 | 6 | 2 | 2 | 2 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 5 | 5 | 5 | 8 | 8 |
| frame3 | | | 9 | 9 | 9 | 9 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 7 | 7 | 7 | 5 |
| MISS | Y | Y | Y | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | N | Y | Y | Y | Y | Y | Y |

- 0; 0,6; 0,6,3; 6,3,2; 3,2,5; 2,5,4; 5,4,1; 4,1,3; 1,3,0; 3,0,6; 0,6,1; 6,1,4; 1,4,2; 4,2,3; 2,3,5; 3,5,7(**16 page faults**)

| | **0** | **6** | **3** | **0** | **2** | **6** | **3** | **5** | **2** | **4** | **1** | **3** | **0** | **6** | **1** | **4** | **2** | **3** | **5** | **7** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 6 | 6 | 6 | 2 | 2 | 2 | 7 |
| frame2 | | 6 | 6 | 6 | 6 | 6 | 6 | 5 | 5 | 5 | 5 | 3 | 3 | 3 | 1 | 1 | 1 | 3 | 3 | 3 |
| frame3 | | | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 0 | 0 | 0 | 4 | 4 | 4 | 5 | 5 |
| MISS | Y | Y | Y | N | Y | N | N | Y | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |

- 3; 3,1; 3,1,4; 1,4,2; 4,2,5; 2,5,1; 5,1,3; 1,3,2; 3,2,0; 2,0,1; 0,1,3; 1,3,4; 3,4,5; 4,5,0; 5,0,1(**15 page faults**)

| | **3** | **1** | **4** | **2** | **5** | **4** | **1** | **3** | **5** | **2** | **0** | **1** | **1** | **0** | **2** | **3** | **4** | **5** | **0** | **1** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 5 | 5 | 5 |
| frame2 | | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 5 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 0 | 0 |
| frame3 | | | 4 | 4 | 4 | 4 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 1 |
| MISS | Y | Y | Y | Y | Y | N | Y | Y | N | Y | Y | Y | N | N | N | Y | Y | Y | Y | Y |

- 4; 4,2; 4,2,1; 2,1,7; 1,7,9; 7,9,8; 9,8,3; 8,3,5; 3,5,2; 5,2,6; 2,6,8; 6,8,1; 8,1,0; 1,0,7; 0,7,2; 7,2,4; 2,4,1; 4,1,3; 1,3,5; 3,5,8(**20 page faults**)

| | **4** | **2** | **1** | **7** | **9** | **8** | **3** | **5** | **2** | **6** | **8** | **1** | **0** | **7** | **2** | **4** | **1** | **3** | **5** | **8** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 4 | 4 | 4 | 7 | 7 | 7 | 3 | 3 | 3 | 6 | 6 | 6 | 0 | 0 | 0 | 4 | 4 | 4 | 5 | 5 |
| frame2 | | 2 | 2 | 2 | 9 | 9 | 9 | 5 | 5 | 5 | 8 | 8 | 8 | 7 | 7 | 7 | 1 | 1 | 1 | 8 |
| frame3 | | | 1 | 1 | 1 | 8 | 8 | 8 | 2 | 2 | 2 | 1 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 |
| MISS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |

- 0; 0,1; 0,1,2; 1,2,3; 2,3,4; 3,4,1; 4,1,0; 1,0,2; 0,2,3; 2,3,4; 3,4,1; 4,1,0(**12 page faults**)

| | **0** | **1** | **2** | **3** | **4** | **4** | **3** | **2** | **1** | **0** | **0** | **1** | **2** | **3** | **4** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 4 | 4 | 4 |
| frame2 | | 1 | 1 | 1 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 |
| frame3 | | | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 0 |
| MISS | Y | Y | Y | Y | Y | N | N | N | Y | Y | N | N | Y | Y | Y | N | N | N | Y | Y |

**LRU:**

- 2; 2,6; 2,6,9; 2,4,9; 2,4,1; 2,7,1; 1,7,3; 7,3,0; 3,0,5; 0,5,2; 5,2,1; 9,2,1; 9,2,5; 9,5,7; 5,7,3; 7,3,8; 3,8,5(**17 page faults**)

| | **2** | **6** | **9** | **2** | **4** | **2** | **1** | **7** | **3** | **0** | **5** | **2** | **1** | **2** | **9** | **5** | **7** | **3** | **8** | **5** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 | 5 |
| frame2 | | 6 | 6 | 6 | 4 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 5 | 9 | 9 | 9 | 3 | 3 | 3 |
| frame3 | | | 9 | 9 | 9 | 9 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 5 | 5 | 5 | 8 | 8 |
| MISS | Y | Y | Y | N | Y | N | Y | Y | Y | Y | Y | Y | Y | N | Y | Y | Y | Y | Y | Y |

- 0; 0,6; 0,6,3; 3,0,2; 0,2,6; 2,6,3; 6,3,5; 3,5,2; 5,2,4; 2,4,1; 4,1,3; 1,3,0; 3,0,6; 0,6,1; 6,1,4; 1,4,2; 4,2,3; 2,3,5; 3,5,7(**19 page faults**)

| | **0** | **6** | **3** | **0** | **2** | **6** | **3** | **5** | **2** | **4** | **1** | **3** | **0** | **6** | **1** | **4** | **2** | **3** | **5** | **7** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 4 | 4 | 4 | 0 | 0 | 0 | 4 | 4 | 4 | 5 | 5 |
| frame2 | | 6 | 6 | 6 | 2 | 2 | 2 | 5 | 5 | 5 | 1 | 1 | 1 | 6 | 6 | 6 | 2 | 2 | 2 | 7 |
| frame3 | | | 3 | 3 | 3 | 6 | 6 | 6 | 2 | 2 | 2 | 3 | 3 | 3 | 1 | 1 | 1 | 3 | 3 | 3 |
| MISS | Y | Y | Y | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |

- 3; 3,1; 3,1,4; 1,4,2; 4,2,5; 5,4,1; 4,1,3; 1,3,5; 3,5,2; 5,2,0; 2,0,1; 0,2,3; 2,3,4; 3,4,5; 4,5,0; 5,0,1(**16 page faults**)

| | **3** | **1** | **4** | **2** | **5** | **4** | **1** | **3** | **5** | **2** | **0** | **1** | **1** | **0** | **2** | **3** | **4** | **5** | **0** | **1** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 3 | 3 | 3 | 2 | 2 | 2 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 5 | 5 | 5 |
| frame2 | | 1 | 1 | 1 | 5 | 5 | 5 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 1 |
| frame3 | | | 4 | 4 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 1 | 1 | 1 | 1 | 3 | 3 | 3 | 0 | 0 |
| MISS | Y | Y | Y | Y | Y | N | Y | Y | Y | Y | Y | Y | N | N | N | Y | Y | Y | Y | Y |

- 4; 4,2; 4,2,1; 2,1,7; 1,7,9; 7,9,8; 9,8,3; 8,3,5; 3,5,2; 5,2,6; 2,6,8; 6,8,1; 8,1,0; 1,0,7; 0,7,2; 7,2,4; 2,4,1; 4,1,3; 1,3,5; 3,5,8(**20 page faults**)

| | **4** | **2** | **1** | **7** | **9** | **8** | **3** | **5** | **2** | **6** | **8** | **1** | **0** | **7** | **2** | **4** | **1** | **3** | **5** | **8** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 4 | 4 | 4 | 7 | 7 | 7 | 3 | 3 | 3 | 6 | 6 | 6 | 0 | 0 | 0 | 4 | 4 | 4 | 5 | 5 |
| frame2 | | 2 | 2 | 2 | 9 | 9 | 9 | 5 | 5 | 5 | 8 | 8 | 8 | 7 | 7 | 7 | 1 | 1 | 1 | 8 |
| frame3 | | | 1 | 1 | 1 | 8 | 8 | 8 | 2 | 2 | 2 | 1 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 |
| MISS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |

- 0; 0,1; 0,1,2; 1,2,3; 2,3,4; 3,2,1; 2,1,0; 1,2,3; 2,3,4; 3,2,1; 2,1,0(**11 page faults**)

| | **0** | **1** | **2** | **3** | **4** | **4** | **3** | **2** | **1** | **0** | **0** | **1** | **2** | **3** | **4** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 0 |
| frame2 | | 1 | 1 | 1 | 4 | 4 | 4 | 4 | 1 | 1 | 1 | 1 | 1 | 1 | 4 | 4 | 4 | 4 | 1 | 1 |
| frame3 | | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| MISS | Y | Y | Y | Y | Y | N | N | N | Y | Y | N | N | N | Y | Y | N | N | N | Y | Y |

**optimal (OPT):**

- 2; 2,6; 2,6,9; 2,4,9; 2,1,9; 1,2,7; 1,2,3; 1,2,0; 1,2,5; 9,2,5; 9,5,7; 5,7,3; 5,8,3(**13 page faults**)

| | **2** | **6** | **9** | **2** | **4** | **2** | **1** | **7** | **3** | **0** | **5** | **2** | **1** | **2** | **9** | **5** | **7** | **3** | **8** | **5** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 9 | 9 | 7 | 3 | 8 | 8 |
| frame2 | | 6 | 6 | 6 | 4 | 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| frame3 | | | 9 | 9 | 9 | 9 | 9 | 7 | 3 | 0 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| MISS | Y | Y | Y | N | Y | N | Y | Y | Y | Y | Y | N | N | N | Y | N | Y | Y | Y | N |

- 0; 0,6; 0,6,3; 6,3,2; 2,3,5; 2,3,4; 1,3,4; 1,4,0; 1,4,6; 1,4,2; 2,3,4; 2,3,5; 3,5,7(**13 page faults**)

| | **0** | **6** | **3** | **0** | **2** | **6** | **3** | **5** | **2** | **4** | **1** | **3** | **0** | **6** | **1** | **4** | **2** | **3** | **5** | **7** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 3 | 5 | 7 |
| frame2 | | 6 | 6 | 6 | 6 | 6 | 6 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| frame3 | | | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| MISS | Y | Y | Y | N | Y | N | N | Y | N | Y | Y | N | Y | Y | N | N | Y | Y | Y | Y |

- 3; 3,1; 3,1,4; 1,4,2; 1,4,5; 1,3,5; 1,2,3; 0,1,2; 0,1,3; 0,1,4; 0,1,5(**11 page faults**)

| | **3** | **1** | **4** | **2** | **5** | **4** | **1** | **3** | **5** | **2** | **0** | **1** | **1** | **0** | **2** | **3** | **4** | **5** | **0** | **1** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 3 | 3 | 3 | 2 | 5 | 5 | 5 | 5 | 5 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 4 | 5 | 5 | 5 |
| frame2 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| frame3 | | | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MISS | Y | Y | Y | Y | Y | N | N | Y | N | Y | Y | N | N | N | N | Y | Y | Y | N | N |

- 4; 4,2; 4,2,1; 2,1,7; 2,1,9; 2,1,8; 2,3,8; 2,5,8; 2,6,8; 2,1,8; 2,1,0; 2,1,7; 2,1,4; 3,4,1; 3,5,1; 3,5,8(**16 page faults**)

| | **4** | **2** | **1** | **7** | **9** | **8** | **3** | **5** | **2** | **6** | **8** | **1** | **0** | **7** | **2** | **4** | **1** | **3** | **5** | **8** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 4 | 4 | 4 | 7 | 9 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 0 | 7 | 7 | 4 | 4 | 3 | 5 | 8 |
| frame2 | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| frame3 | | | 1 | 1 | 1 | 1 | 3 | 5 | 5 | 6 | 6 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| MISS | Y | Y | Y | Y | Y | Y | Y | Y | N | Y | N | Y | Y | Y | N | Y | N | Y | Y | Y |

- 0; 0,1; 0,1,2; 1,2,3; 2,3,4; 1,2,3; 0,1,2; 1,2,3; 2,3,4; 1,2,3; 0,1,2(**11 page faults**)

| | **0** | **1** | **2** | **3** | **4** | **4** | **3** | **2** | **1** | **0** | **0** | **1** | **2** | **3** | **4** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| frame1 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 1 | 1 |
| frame2 | | 1 | 1 | 1 | 4 | 4 | 4 | 4 | 1 | 1 | 1 | 1 | 1 | 1 | 4 | 4 | 4 | 4 | 4 | 4 |
| frame3 | | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| MISS | Y | Y | Y | Y | Y | N | N | N | Y | Y | N | N | N | Y | Y | N | N | N | Y | Y |

## 10.35

**Question:** A page-replacement algorithm should minimize the number of page faults. We can achieve this minimization by distributing heavily used pages evenly over all of memory, rather than having them compete for a small number of page frames. We can associate with each page frame a counter of the number of pages associated with that frame. Then, to replace a page, we can search for the page frame with the smallest counter.

a. Define a page-replacement algorithm using this basic idea. Specifically address these problems:
  • What is the initial value of the counters?
  • When are counters increased?
  • When are counters decreased?
  • How is the page to be replaced selected?

b. How many page faults occur for your algorithm for the following reference string with four page frames?
  1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2.

c. What is the minimum number of page faults for an optimal page replacement strategy for the reference string in part b with four page frames?

**Answer:**

a. Answer these question one by one.

- The initial value of the counter is ZERO(0);
- When a new page is associated with that frame, in other word, when this frame is required by a new page, the counters will be increased.
- When one of the pages which are associated to that frame is not required anymore, then the counters will be decreased.
- System will find a frame which hold the smallest counter, and then use FIFO to replace the page.

b. We can count that in the table there are 14 "M"(miss), which means that there are 14 page faults.

| **Frame Number** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1(M) | 5(M) | 5(H) | 5(H) | | | | | |
| 2 | 2(M) | 8(M) | 8(H) | 7(M) | 9(M) | | | | |
| 3 | 3(M) | 3(H) | 1(M) | 6(M) | 7(M) | 7(H) | 9(M) | 8(M) | 2(M) |
| 4 | 4(M) | 4(H) | 4(H) | 4(H) | | | | | |

c. We can count that in the table there are 11 "M"(miss), which means that there are 11 page faults.

| **Frame Number** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1(M) | 1(H) | 6(M) | 8(M) | 8(H) | 8(H) | 4(M) | 4(H) | 2(M) |
| 2 | 2(M) | 5(M) | 5(H) | 5(H) | | | | | |
| 3 | 3(M) | 3(H) | 7(M) | 7(H) | 7(H) | | | | |
| 4 | 4(M) | 4(H) | 9(M) | 9(H) | | | | | |

## 10.36

**Question:** Consider a demand-paging system with a paging disk that has an average access and transfer time of 20 milliseconds. Addresses are translated through a page table in main memory, with an access time of 1 microsecond per memory access. Thus, each memory reference through the page table takes two accesses. To improve this time, we have added an associative memory that reduces access time to one memory reference if the page-table entry is in the associative memory. Assume that 80 percent of the accesses are in the associative memory and that, of those remaining, 10 percent (or 2 percent of the total) cause page faults. What is the effective memory access time?

**Answer:** The effective memory access time is $401.2\mu s$, as the following calculation shown.

$$\begin{aligned} effective\ memory\ access\ time\ &= (80\% * 1\mu s) + (18\% * 2 * 1\mu s) + (2\% * (20ms + 2 * 1\mu s)) \\ &= 0.8\mu s + 0.36\mu s + 400.04\mu s \\ &= 401.2\mu s \end{aligned}$$

## 10.40

**Question:** In a 1,024-KB segment, memory is allocated using the buddy system. Using Figure 10.26 as a guide, draw a tree illustrating how the following memory requests are allocated:

- Request 5-KB
- Request 135 KB.
- Request 14 KB.
- Request 3 KB
- Request 12 KB.

Next, modify the tree for the following releases of memory. Perform coalescing whenever possible:

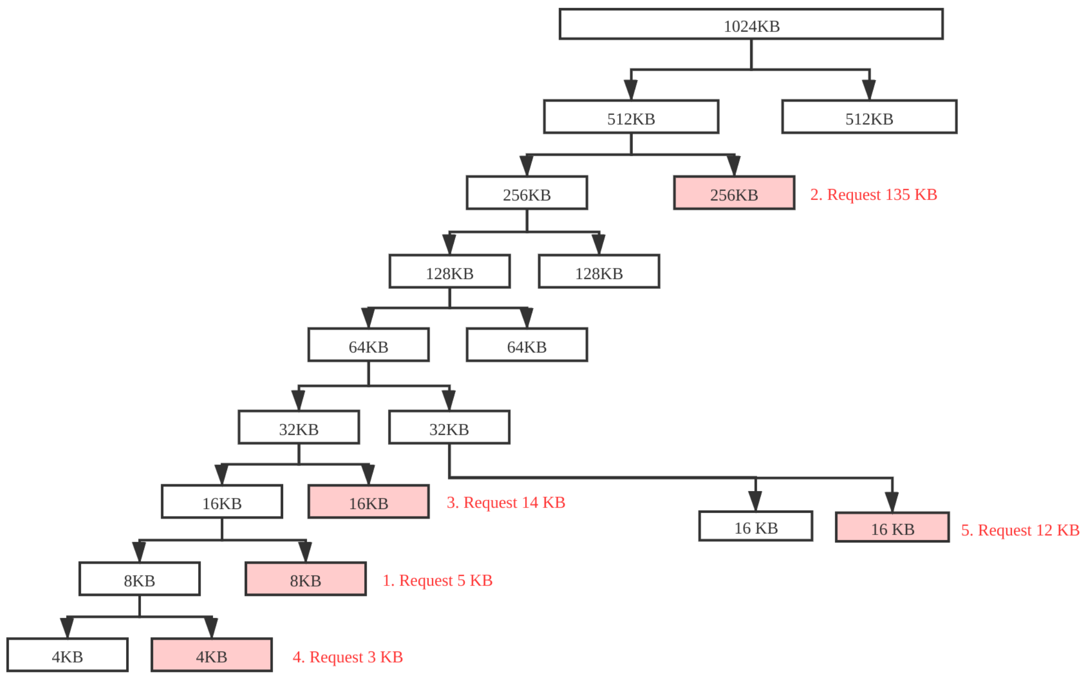
- Release 3 KB.
- Release 5 KB.
- Release 14 KB.
- Release 12 KB.

**Answer:**

**solve requests:**

The following allocation is made by the Buddy system: The 5-KB request is assigned a 8-KB segment. The 135-KB request is assigned a 256-KB segment. The 14-KB request is assigned a 16-KB segment, the 3-KB request is assigned a 4-KB segment and the 12-KB request is assigned a 16-KB segment .

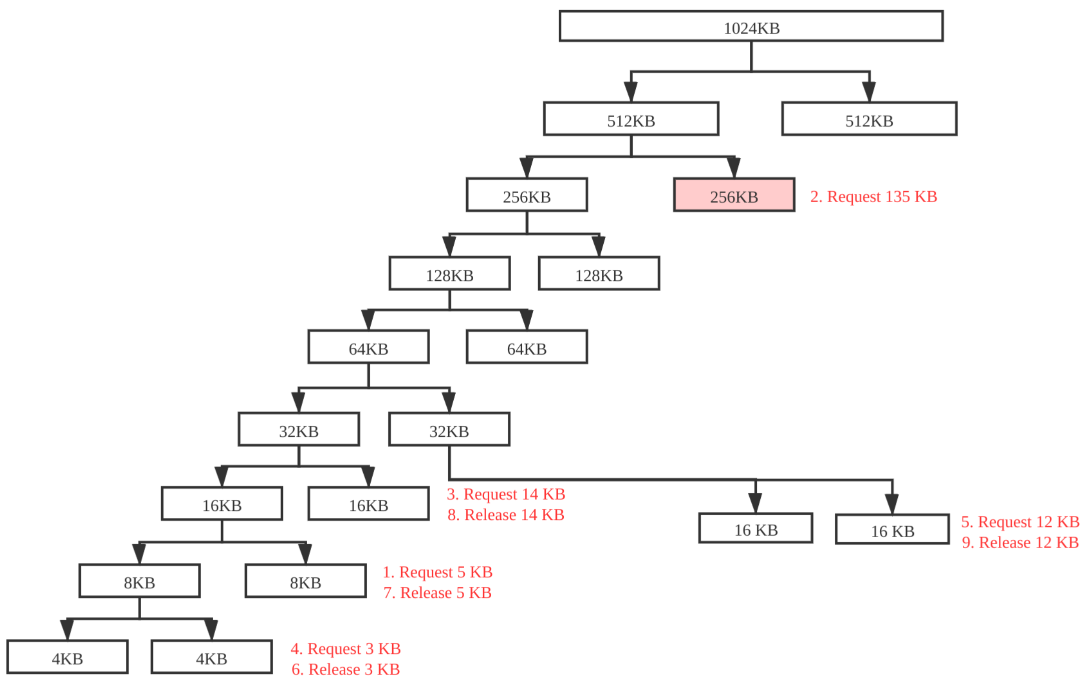
After all the memory requests are allocated, the tree structure illustration shown below.



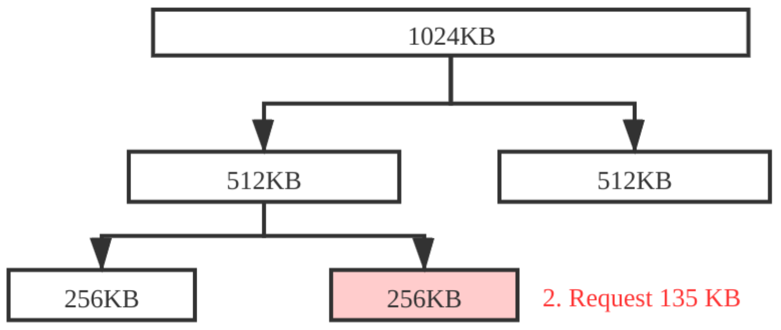
**solve releases:**

After the releases of memory, the only segment in use would be a 256-KB segment containing 135 bytes of data, and the tree structure illustration shown below.

This is the version I do not have complete merge operation.(**Attention:** this version is just for making you understand easily.)



This is the merged version.



1024KB
512KB
512KB
256KB
256KB
2. Request 135 KB