8

8.3

a. First-fit:

1) 212K is put in 500K partition

2) 417K is put in 600K partition

3) 112K is put in 288K partition (new partition 288K = 500K - 212K)

4) 426K must wait

b. Best-fit:

1) 212K is put in 300K partition

2) 417K is put in 500K partition

3) 112K is put in 200K partition

4) 426K is put in 600K partition

c. Worst-fit:

1) 212K is put in 600K partition

2) 417K is put in 500K partition

3) 112K is put in 388K partition

4) 1 426K must wait

In this example, Best-fit turns out to be the best.

8.5

	External fragmentation	Internal fragmentation	Ability to share code across processes
Contiguous	Yes	No	Difficult
Pure segmentation	Yes	No	Easy most
Pure paging	No	Yes	easy

8.9

a. 400 nanoseconds; 200 nanoseconds to access the page table and 200 nanoseconds to access the word in memory.

b. Effective access time = 0.75 × (200 nanoseconds) + 0.25× (400 nanoseconds) = 250 nanoseconds.

8.12

- **219+430=649**
- **2300+10=2310**
- **Error**
- **1327+400=1727**
- **error**

9

9.4

- **Virtual memory space: 2^{32}**
- **Physical memory: 2^{18} B**
- **Page size: 4096B=2^{12} B**
- **Logical address: 11123456**
- **Translation:**
  - **11123456/4096: S=2715 R=2816**
  - **Page number p=2715 offset d=2816**
  - **Page table[2715]=frame number F**
  - **Physical address: F d**

9.5

0.2μsec = (1-p)*0.1μsec + (0.3p)*8 millisec +(0.7p)*20 millisec

0.1= -0.1p+2400p+14000p

0.1 ≈ 16,400p

P ≈ 0.000006

9.11

- **Page fault 1** 把指令 **load [M]**所在页调入内存
- **Page fault 2** 把地址 **M** 所在页面调入内存，取出其内容 **a**
- **Page fault 3** 将地址 **a** 对应的页面调入内存，从中取出内容
- **Thrashing**

10

10.2

- **Maintain one table that contains references to files that are being accessed by all users at the current time.**
- **One entry with shared counter.**

10.9

- **One copy:** 便于保持数据的一致性和完整性，操作互斥
- **More copy:** 维护数据的一致性比较困难

11

11.1

- 分配快，无外部碎片，有内部碎片
- 分配慢，有外部碎片，无内部碎片
- 介于 **a,b** 之间

11.2

支持随机访问

11.3

a. In order to reconstruct the free list, it would be necessary to perform "garbage collection." This would entail searching the entire directory structure to determine which pages were

already allocated to jobs. Those remaining unallocated pages could be relinked as the free-space list.

c. The free-space list pointer could be stored on the disk, perhaps in several places.

11.6

Let *Z* be the starting file address (block number).

a. Contiguous. Divide the logical address by 512 with *X* and *Y* the resulting quotient and remainder respectively.

i. Add *X* to *Z* to obtain the physical block number. *Y* is the displacement into that block.

ii. 1

b. Linked. Divide the logical physical address by 511 with *X* and *Y* the resulting quotient and remainder respectively.

i. Chase down the linked list (getting *X* + 1 blocks). *Y* + 1 is the displacement into the last physical block.

ii. 4

c. Indexed. Divide the logical address by 512, with *X* and *Y* the resulting quotient and remainder, respectively.

i. Get the index block into memory. Physical block address is contained in the index block at location *X*. *Y* is the displacement into the desired physical block.

ii. 2

12

12.1

a. New requests for the track over which the head currently resides can theoretically arrive as quickly as these requests are being serviced.

b. All requests older than some predetermined age could be "forced" to the top of the queue, and an associated bit for each could be set to indicate that no new request could be moved ahead of these requests. For SSTF, the rest of the queue would have to be reorganized with respect to the last of these "old" requests.

c. To prevent unusually long response times.

d. Paging and swapping should take priority over user requests. It may be desirable for other kernel-initiated I/O, such as the writing of file system metadata, to take precedence over user I/O. If the kernel supports real-time process priorities, the I/O requests of those processes should be favored.

12.2

a. The FCFS schedule is 143, 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. The total seek distance is 7081.

b. The SSTF schedule is 143, 130, 86, 913, 948, 1022, 1470, 1509, 1750, 1774. The total seek distance is 1745.

c. The SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 4999, 130, 86. The total seek distance is 9769.

d. The LOOK schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 130, 86. The total seek

distance is 3319.

e. The C-SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 4999, 86, 130. The total seek distance is 9985.

f. (Bonus.) The C-LOOK schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 86, 130. The total seek distance is 3363.

13

13.6

Generally, blocking I/O is appropriate when the process will only be waiting for one specific event. Examples include a disk, tape, or keyboard read by an application program. Non-blocking I/O is useful when I/O may come from more than one source and the order of the I/O arrival is not predetermined. Examples include network daemons listening to more than one network socket, window managers that accept mouse movement as well as keyboard input, and I/O-management programs, such as a copy command that copies data between I/O devices. In the last case, the program could optimize its performance by buffering the input and output and using non-blocking I/O to keep both devices fully occupied.

Non-blocking I/O is more complicated for programmers, because of the asynchronous rendezvous that is needed when an I/O occurs. Also, busy waiting is less efficient than interrupt-driven I/O so the overall system performance would decrease.