# Sample Solution of Assignment 4:

Part I:

1. No sample solution. You need to discuss complications of the concurrent processing on OS design, e.g., context switching, process synchronization, process scheduling, memory management, deadlock prevention, interprocess communication, and much more. Since there are many things you can mention, this question is essentially a take-it-for-free question, unless your discussion is nonsense or equal to saying nothing (e.g., handling concurrency is much more complicated).

2. (a)
10 13 10 15
10 15 10 13

(b)
(1) add *wait(NULL)* before *i+=5*
(2) add *ffush()* before *if (fork())*

(Note that (1) is the must in order to get the score)

Part II.

3. The signal() operation associated with monitors is not persistent in the following sense: if a signal is performed and if there are no waiting threads, then the signal is simply ignored and the system does not remember that the signal took place. If a subsequent wait operation is performed, then the corresponding thread simply blocks. In semaphores, on the other hand, every signal results in a corresponding increment of the semaphore value even if there are no waiting threads. A future wait operation would immediately succeed because of the earlier increment.

4.

```
monitor printers {
        int num avail = 3;
        int waiting processes[MAX PROCS];
        int num waiting;
        condition c;
        void request printer(int proc number) {
                if (num avail > 0) {
                num avail--;
                return;
                }
                waiting processes[num waiting] = proc number;
                num waiting++;
                sort(waiting processes);
                while (num avail == 0 &&
                        waiting processes[0] != proc number)
                        c.wait();
                waiting processes[0] =
                waiting processes[num waiting-1];
                num waiting--;
                sort(waiting processes);
                num avail--;
        }

        void release printer() {
                num avail++;
                c.broadcast();
        }
}
```

5. This is the typical sleeping-barbar problem. One sample solution could be found at:

http://caig.cs.nctu.edu.tw/course/OS10/OS_SleepingBarber_F10.pdf

Part III.

6.

a)

The maximum transfer rate depends on the size of the track. For maximum efficiency we look at the tracks with the most sectors. In the case of Rhinopias, this is a track with 1000 sectors. Assuming a 10,000 RPM rotational speed, we can get 166.67 rotations per second (10,000/60 = ~166.67). For the single track, with 1000 sectors, where there are 512 bytes per sector, this yields 166.67 * 512000 = 85.335 MB per second.

b)

When we access multiple tracks, the delay in moving the head to the different tracks must be factored in. Assuming head skewing, there is a delay of one sector in order to get each subsequent track. If we look at the tracks with the most sectors (the outer tracks) then the rotational delay to move one sector is 1/1000 of a disk rotation time. A disk rotation time is 0.006 seconds, so 0.006/1000 = 0.000006 seconds (or 6 microseconds). We then add 6 microseconds for each track but the last, yielding an additional 24 microseconds. Previously, we could transfer one track (512,000 bytes) in 0.000006 * 1000 seconds. This time is now (0.000006 * 1000) + 0.000006 seconds.

This yields 166.5 * 512000 = 85.248 MB/sec maximum transfer rate.

7.

In the early 1980s the cost of primary storage was so high that one could not afford enough of it for a cache that would be sufficiently large to hold enough recently read file-system blocks to mitigate the cost of reading them from disk.

8.

To rename a file you should first create a link to the file under its new name, then remove the link corresponding to its old name. Creating the new link cannot be done atomically, since both the directory must be updated to contain the link and the file's inode must be updated to increase its link count from 1 to 2. If the directory entry is created before the link count is updated, there could be a problem if the system crashes between the two operations. After the crash, though there are two links to the file, its link count is one. If one of the links is removed, the link count is reduced to zero and the file might be deleted since it appears to be no longer referenced. On the other hand, if the link count is updated before the directory entry is created, and a crash occurs between the two operations, there will be just one entry referring to the file, even though it has a link count of two. The worst that could happen in this case is that someone intends to delete the file by removing that link, but because it would still have a positive link count, the file continues to exist, but is not referenced by any directory. This, as explained in the text, is an innocuous inconsistency.

After the new link is created, we have a similar problem in removing the old link. Should the link be removed first, then the file's link count decremented by one, or vice versa? Following the reasoning similar to the above, we should first remove the old link, then decrement the file's link count, so that the link count is always at least as great as the actual number of links.

9. No sample solution. This is an open question. For a good recommendation, you need to consider the limited disk space, the difficulty in supporting variable length of entries in directory, and so on.

Part IV.

10.
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.

11. a. First-fit:
- 212K is put in 500K partition
- 417K is put in 600K partition
- 112K is put in 288K partition (new partition 288K = 500K − 212K)
- 426K must wait

b. Best-fit:
- 212K is put in 300K partition

417K is put in 500K partition
112K is put in 200K partition
426K is put in 600K partition

c.Worst-fit:

212K is put in 600K partition
417K is put in 500K partition
112K is put in 388K partition
426K must wait

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

12.

a. page = 1; offset = 391
b. page = 18; offset = 934
c. page = 29; offset = 304
d. page = 0; offset = 256
e. page = 1; offset = 1

13. Read the reference book (the so-called dinosaur book, Operating System Concepts, 8th edition, by Silberschatz et al.) Chapter 8.5.3. The book is reserved in the library.

14. If the page number is in TLB, then the access time is 3ns. If the page number is not in TLB, then it needs 1ns for TLB lookup plus 10 ns for one word reading via the page table. Denote the TLB hit rate as h, then $3*h + 11 * (1-h) <= 5$. We have $h >= 75\%$