The quiz of this week required much of searching through these sections in the book, which was quite tedious and I can only memorize few of them. The list below are things I considered noteworthy after reading the book again for the essay:

APUE 3.09 I/O Efficiency

All normal UNIX system shells provide a way to open a file for reading on standard  
input and to create (or rewrite) a file on standard output. This prevents the  
program from having to open the input and output files, and allows the user to  
take advantage of the shell’s I/O redirection facilities.

Most file systems support some kind of read-ahead to improve performance. When  
sequential reads are detected, the system tries to read in more data than an application  
requests, assuming that the application will read it shortly. The effect of read-ahead can  
be seen in Figure 3.6, where the elapsed time for buffer sizes as small as 32 bytes is as  
good as the elapsed time for larger buffer sizes.

APUE 3.10 File Sharing

The kernel uses three data structures to represent an open file, and the relationships  
among them determine the effect one process has on another with regard to file sharing.  
1. Every process has an entry in the process table. Within each process table entry is a  
table of open file descriptors, which we can think of as a vector, with one entry per  
descriptor. Associated with each file descriptor are  
(a) The file descriptor flags (close-on-exec; refer to Figure 3.7 and Section 3.14)  
(b) A pointer to a file table entry  
2. The kernel maintains a file table for all open files. Each file table entry contains  
(a) The file status flags for the file, such as read, write, append, sync, and  
nonblocking  
(b) The current file offset  
(c) A pointer to the v-node table entry for the file  
3. Each open file (or device) has a v-node structure that contains information about the  
type of file and pointers to functions that operate on the file. For most files, thev-node also contains the i-node for the file. This information is read from disk when  
the file is opened, so that all the pertinent information about the file is readily  
available. For example, the i-node contains the owner of the file, the size of the file,  
pointers to where the actual data blocks for the file are located on disk, and so on

APUE 3.11 Atomic Operations

• After each write is complete, the current file offset in the file table entry is  
incremented by the number of bytes written. If this causes the current file offset  
to exceed the current file size, the current file size in the i-node table entry is set  
to the current file offset (for example, the file is extended).  
• If a file is opened with the O\_APPEND flag, a corresponding flag is set in the file  
status flags of the file table entry. Each time a write is performed for a file with  
this append flag set, the current file offset in the file table entry is first set to the  
current file size from the i-node table entry. This forces every write to be  
appended to the current end of file.  
• If a file is positioned to its current end of file using lseek, all that happens is the  
current file offset in the file table entry is set to the current file size from the  
i-node table entry. (Note that this is not the same as if the file was opened with  
the O\_APPEND flag, as we will see in Section 3.11.)  
• The lseek function modifies only the current file offset in the file table entry.  
No I/O takes place.

It is possible for more than one file descriptor entry to point to the same file table  
entry

APUE 3.12 dup and dup2 functions

The new file descriptor returned by dup is guaranteed to be the lowest-numbered  
available file descriptor. With dup2, we specify the value of the new descriptor with the  
*fd2* argument. If *fd2* is already open, it is first closed. If *fd* equals *fd2*, then dup2 returns  
*fd2* without closing it. Otherwise, the FD\_CLOEXEC file descriptor flag is cleared for *fd2*,  
so that *fd2* is left open if the process calls exec.