Group A

1. CPU utilization = 1-pn, where p is I/O Delay, and n is the number of process in memory.  
   a) 80% delay, n = 2  
   1-0.82 = 0.36 / 2 = 0.18 per process. 10 minutes / 0.18 = 55.5 minutes  
   b) 90 % delay, n = 2  
   1-0.92 = 0.19 / 2 = 0.095 per process. 10 minutes / 0.095 = 105.3 minutes.
2. Single threaded (.85 \* 10) + (.15 \* 60) = 17.5  
   Multithreaded is always 100/sec, as the CPU does not have to wait.

Group B

1. Figure 2-2 does not show transitions for moving from ready to blocked, or from blocked to running. It can receive the input it is looking for and will continue running.
2. A portion of all interrupt handlers are written in assembly to provide direct access to CPU registers.
3. For protection. It keeps the kernel address space free from user space data, which may have many problems, unintentional (unflushed data) or otherwise. For example, most Wii exploits work when malicious code is copied into the address space of a running process, allowing the programmer to gain control (supervisor mode) of the system.
4. No, a single threaded process can fork while blocked, because it is blocked and can’t do anything.
5. User-level; the kernel must be able to free the blocking read calls often to service many requests.
6. In the case of a multithreaded process, a thread may yield to another in its process to allow it to do some important thing.
7. Yes, in the case of a time-based scheduler, a timer may expire and the kernel will issue an interrupt to hand control to another thread.
8. The process should issue pthread\_join NUMBER\_OF\_THREADS on each iteration of I to force threads to wait for the previous one before starting.
9. Yes, the system is possible. Setting the clock like a timeout for processes would unblock them if they are waiting due to read failure. However, it would also be required to unset the clock in this application.