Group A

1. The fundamental difference between kernel mode and user mode is the level of access to resources in each mode. Processes running in kernel mode can access all resources, while processes running within user mode cannot.
2. The two main ideas of operating systems focus on the domain of the functions an OS performs. As a resource manager, the OS must maintain control of space and time resources in order to appropriately divide them among the competing users/programs/etc. For example, an OS spooling print jobs to issue them one-by-one directly to the printer. When acting as an extended machine, the OS serves as an abstraction layer between software and hardware, encasing unfriendly hardware interface in an easier-to-understand format for development of application programs. For example, Windows builds a HAL (Hardware Abstraction Layer) that initializes the disk controller directly before handing control over to user-mode software, allowing the software to use friendly APIs to interface with the disk, rather than the disk directly. One disadvantage of this application is that the HAL must be rebuilt in a lengthy process when the controller is changed.
3. The first generation of computer architecture started with a 300-vacuum tube beast at Iowa State. No operating systems on these machines, just native machine code. The invention of the transistor lead to mainframes where programs were written onto punch cards in FORTRAN and feed into the machine in batches. The third generation of hardware was ushered in with ICs, yielding more performance for less money. Families of computers were introduced here, with cross-machine compatible operating systems allowing programs to run on several different types of computers. Timesharing was developed in this generation, and UNIX began to form, along with the development of the POSIX standard. The last generation is the current one of PCs. The invention of the GUI, along with the decrease in major component size and cost, furthered opening computers to all people. DOS, Linux, Mac OS X, and Windows are some the most recognizable OSes to run in this generation.
4. Both timesharing and multiprogramming are strategies to address the same problem; making the most out of CPU time. Initially, CPUs were very fast and peripherals very slow, so multiple programs would be loaded into memory and run one at a time until a peripheral request was made. Upon this, the waiting program would be cached back to memory and the next program started. However, as computing moved from batches of programs run once at a time toward an interactive environment shared among users, timesharing was developed. Timesharing involved having multiple users on individual terminals interfaced to one machine working with their own accounts, giving the feeling that they are working on the machine alone.
5. I do not think mobile devices will require spooling. Spooling refers to buffering of data to accommodate a contrastingly slower medium. I think the current evolution trend of this technology will lead to fairly synchronous mediums in all aspects.

Group B

1. Multiprogramming would be impossible in this scenario, the CPU would have to completely finish a job before starting another
2. This idea survives—NetBooks are the most recent example of a “family of computers”; a similar group of computers with ranging power that run common software.
3. A,c,d
4. 1 billion
5. [2 ns \* .95] + [10ns \* (.99 \* .05)] + [10000000 \* (.01 \* .05)] ~ 5000 ns
6. A trap instruction is a special type of command that is used to elevate a process from user mode to kernel mode. It is used by user mode software when they need to do something that the OS would do in kernel mode.
7. The key difference between an interrupt and a trap is that interrupts occur asynchronously, where traps occur synchronously. That is, traps occur in its turn like any other instruction, but a interrupt stops the CPU and demand control.