1. **Intel processors have supported dual-mode operation since the 386, switching to kernel mode when executing a trap instruction. Why is it important to the OS that the CPU has this functionality, and why not implement it using a simpler “switch to kernel mode” instruction?**

   *(Note that I am not asking for a complete description of “kernel mode”).*

   User mode is fine for most purposes, but kernel mode is necessary to restrict certain operations (e.g., disable interrupts, write anywhere in memory, access I/O ports) to authorized use only. A simple “switch to kernel mode” instruction will not suffice since that could easily be used by a malicious user.

2. **On a typical Windows or Macintosh computer, most operations can be carried out through the graphical user interface. Why do such systems also supply a command line interface?**

   The CLI may be more convenient for certain tasks, allow easy specification of options at the time the program is invoked, may provide programmability missing in the GUI, or simply may be more comfortable for some users.

3. **Why is it necessary for the OS to distinguish between processes that are ready to run (e.g., in the ready queue) and those that are blocked/sleeping and waiting for an I/O operation to complete?**

   Those that are “ready” are not only waiting to be scheduled onto the CPU, but are “ready to run” and will immediately start executing instructions when dispatched to the CPU. Those that are “blocked” are waiting for something to occur and will not execute any instructions even if dispatched to the CPU — so dispatching them to the CPU will simply waste CPU cycles.

4. **Why is it necessary to store the contents of the CPU registers in the Process Control Block before a context switch?**

   Because an executing process has its state stored in the CPU’s PC, SP, and PSW, and has data stored in the CPU’s general purpose registers. If the OS switches to another process, that process will resume operation at the point it was removed from the CPU and will expect its own information to be in place in the CPU. Thus CPU information must be stored in the PCB when a process is removed from the CPU, and restored when a process is dispatched onto the CPU.
5. If a UNIX system forks a child process, that child process gets its own Process Control Block and memory space. Explain why this is necessary, instead of simply sharing these with the parent process.

The child process needs its own PCB because it will have a different process ID, different memory space, different register values, etc. The child process needs its own memory space because, though it will have the same code as its parent, it may execute differently due to different inputs, and thus it will create different internal data (e.g., different data in static variables, a different stack, a different heap for dynamically created objects). Further, the child process could replace its program with a totally different program, in which case there would be little to share with the parent process.