Q1. Why do we say that an Operating System implements an Abstraction? List one good and one bad reason for employing such an abstraction.

An operating system implements an abstraction of the real hardware. This allows a user/programmer to not have to deal with the specifics of the hardware implementation on a particular machine. However, we pay a price in the overhead that is required to support the abstraction.

True or False questions. Justify/explain your answer in one or two sentences:

Q2. Traps are a common mechanism used by an OS to implement the following:

- System call: True. This trap allows for a process in user space to hand control back to the kernel in order to have the kernel do some work on its behalf.

- Page fault: True. This trap is raised when a process accesses a region of its virtual memory that is currently not mapped to physical memory. The trap gives the operating system the opportunity to swap the necessary data from the disk into memory.
• Illegal memory access: True. This trap signifies that a user-level process has attempted to read from / write to memory that it is not allowed to access. This trap allows the OS to decide whether to halt the process or to take some other action.

• Variable assignment: False (most of the time). In general, a variable assignment implies a copy of data between registers and memory, which does not usually raise a trap.

Q3. The context switch overhead of user-level threads is more than that for processes.
False. A context switch is implemented by the kernel and involves the copying of state information between the processor and the Process Control Block or Thread Control Block. Because the kernel knows nothing about user-level threads, there cannot technically be a context switch for user-level threads. Note that the user-level scheduler may perform some limited copying of state on behalf of a thread before control is handed to that thread. However, this copy of state information is much smaller than that of a kernel-level process.

Q4. In a preemptive priority-driven processor, while the processor is busy executing the original job, a new job can enter the processor.
In the literal sense: False. In order for a new job to be given access to the processor, the kernel must have current control of the processor.

In the more general sense: True. Because the scheduler policy is preemptive, the current process may be interrupted and replaced by another process.