1.14.3: What is the purpose of interrupts? How does an interrupt differ from a trap? Can traps be generated intentionally by a user program? If so, for what purpose?

Interrupts allow for signals to be handled asynchronously. Hardware devices are usually one of the things that generate those signals, only when they need the CPU's attention. This allows the CPU to be utilized efficiently across many tasks, dictated by the signals, instead of just waiting for one task to be finished.

An interrupt is an external signal from the hardware devices to the CPU while a trap is an internal signal from a software. An external signal can be masked, but an internal signal can never be masked. One detects the I/O devices and the other detects exceptions or system calls, respectively.

Yes, a user program can intentionally generate a trap via a system call to transition to kernel mode. Its purpose is for assembly coding or x86 coding.

1.14.13: Describe some of the challenges of designing operating systems for mobile devices compared with designing operating systems for traditional PCs.

Power and temperature constraints because the phone is a lot smaller than a PC and is expected to draw less power. Thus, applications should be managed carefully to free up resources and reduce activity when not needed. So, an OS that manages the hardware devices' usage efficiently is needed.

2.13.5: Would it be possible for the user to develop a new command interpreter using the system-call interface provided by the operating system?

Yes. A command interpreter is just a user program that makes use of the OS's system calls.

2.13.11: What is the main advantage of the microkernel approach to system design? How do user programs and system services interact in a microkernel architecture? What are the disadvantages of using the microkernel approach?

A microkernel minimizes running code in privileged mode. Other services like device drivers, filesystems, and network modules are moved to the user-space. So, in the case of a crash in of the following services, they would only affect that particular allocated user-space for that service, not the kernel space.

User programs and system services interact via user requests to the targeted service server, who then receives and constructs a reply message back to the user. This interaction is facilitated by the kernel, who acts as the middleman.

Because user programs and system services must interact via two transitions, user->kernel->service, then service->kernel->user, it can introduce a lot of overhead.