Part 1

1. The input spooler is a buffer for the data that is being read into the computer. The idea of the spoolers whether it was input or output was that the central processing unit (CPU) would be free to do other tasks while a program and/or data was being read or printed. This was done by having dedicated processing units (PU) in the input and output devices.

The card reader processing unit would read a card into the input spooler and then call the Job Control Lanuage interpreter to interpert the information stored in the input spooler. The Job Control Lanuage interperter would parse through the information and decide whether or not it was a special command that the OS would need to know about.

1. The two types of modes in a dual-mode OS are user mode and kernel/privilege mode. This is done by either enabling the mode bit in the CPU for user mode or disabling it for kernel/privilege mode.

User mode is used by the system for processes that the user creates and/or starts in order to ensure that the individual processes have no more control than required over the system's resources/hardware.

Kernel/privilege mode is used only for processes that the system creates and starts as well as for processes that require system's hardware.

The mode changes from user to kernel when a user process invokes a software interrupt to use one of the system's hardwares. After the process is done using the hardware, the software interrupt routine returns, and the mode is changed back to user.

This hardware mechanism is necessary in order to protect the system from malicious users from mismanaging the system's hardware and possibly destroying the system all together.

* 1. The special cards are needed to identify the start of a job, the different sections (text, data, or bss) of the the program, the type of compiler or compilers needed, which libraries to link, call the assembler and finally the end of a job.

The interperter would interpert these special cards after they were loaded into the input spooler. It would interrupt the CPU, and would ask the OS for the appropriate commands.

* 1. In the event of a program crash, the OS would do a core/memory dump of all the contents in memory, CPU state (registers, flags, program counter, stack pointer) and program state just before the crash occured in hexadecimal values. This information was used for debugging and finding why the error occured.

The OS should store information onto the storage device in order for the information to be printed out for analysis.

* 1. **Interactive**: An interactive system is a system that is made up of many short transactions in which the next transaction's result can not be predicted. This system must respond rapidly to user requests as they wait for the results
  2. **Real-time**:A real time system is a system that must process information and respond within a specific time frame or results will be catastrophic.
  3. **Distributed**: A distributed system is a collection of computers that distributes the work equally among the group thus appearing as one enormous computer.
  4. **Parallel**: A parallel system contains multiple processors on dye (chip), multiple CPUs on motherboard or both. Multi-processors on chip usually share the same bus, clock, memory and peripheral devices. Multi-CPUs may share memory and peripeheral devices however they have their own dedicated buses, and clock speeds.

The idea of parallel systems is that there are a number of processors executing processes in parallel.

1. The idea of time sharing was created because of the problem that occured with a single user. A single user would typically mismanage the CPU time allowing the CPU to wait long periods of time. The solution was that it allowed only a specific amount of time to a group of users for the CPU to be used thus reducing the wait time.

This concept was adapted into OS in which the OS would give processes in the ready stage a specific amount of time to run. Once the time was up, the process would be interrupted by the OS, moved back into the ready state, and another process would start running. In the event that a process finished before it's time was up or was waiting for I/O, the process would invoke a software interrupt and the OS would give the CPU to the next process in the ready stage.