Operating Systems in a Distributed Environment: Q & A

By Robert Milton Underwood, Jr.

1. What are the four resources of a virtual computer?

(a) Central processing unit (CPU). CPU scheduling is used to share the CPU with the resources of the physical computer to give the appearance that users have their own processor.
(b) Spooling (simultaneous peripheral operation on-line). Spooling uses the disk as a huge buffer, for reading as far ahead as possible on input devices and for storing output files until the output devices are able to accept them. A spool is, therefore, a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. In some operating systems, spooling is managed by a system daemon process. In other operating systems, it is handled by an in-kernel thread.
(c) File system. It is obvious that a system needs programs that can read and write files.
(d) Time-sharing terminal (provides the function of the virtual machine operator's console)

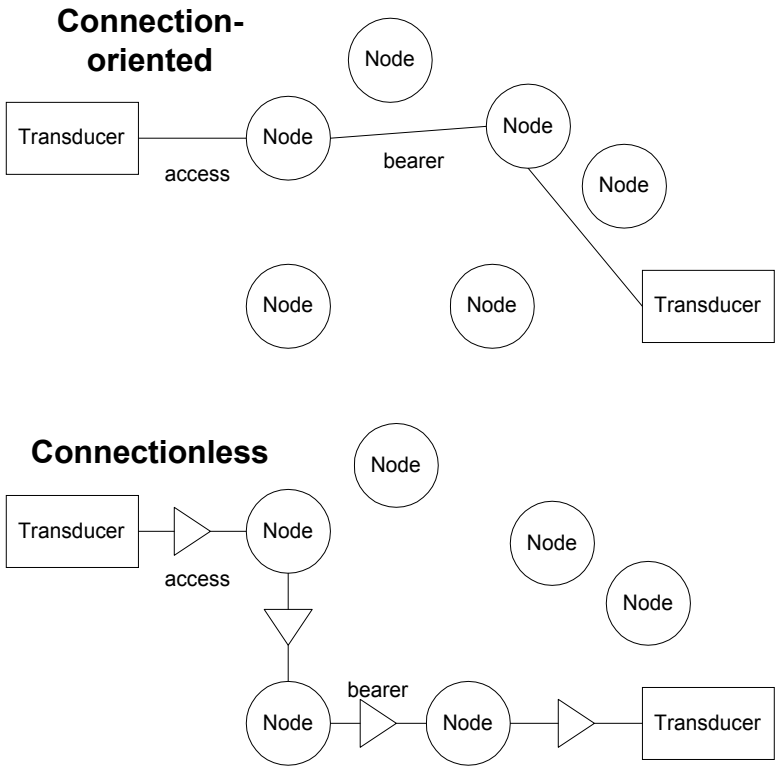
2. What is a connection protocol? What is a connectionless protocol? Give some examples.

A connection-oriented service is a circuit that is established between the sender and receiver (physical - as over a telephone network; or virtual – as over a packet data network). The receiver acknowledges receipt of message segments (packets) and is likely to report the detection of errors. The delay between packets may vary, but packets are delivered in the same sequence in which they were sent. From the point of view of the data, the exchange is simplex; however, actual operation may be duplex or half-duplex, depending on the error-control procedures employed. A connection-oriented protocol is also understood as a term used to describe a communication model that goes through three well-defined stages: (a) establishing the connection, (b) transferring the data, and (c) releasing the connection. It is analogous to an ordinary voice telephone call. Data packets always reach their destination in the same order in which they were sent. Transmission Control Protocol (TCP) is a connection-oriented transport service.

In a connectionless protocol the source, destination, and packet I.D. are included inside each packet so that a direct connection between sender and receiver or an established connection between nodes is not required for communications. In a connectionless service, data packets may not reach their destination in the same order they were sent. (A connectionless message can be unreliable in that the sender is not guaranteed and cannot tell if the packet reached its destination.) The included packet I.D. enables the receiver to put the packets in proper order. Often this service is called datagram service. The User Datagram Protocol (UDP) is one of the specific connectionless transport protocols. Connectionless Network Protocol (CLNP) is an OSI connectionless network protocol. CLNP is the OSI equivalent of Internet Protocol (IP) from the TCP/IP model. A connectionless service is commonly provided over a packet network for short data messages. The receiver sends no acknowledgements. Packets carry originating and terminating addresses. At each network node, they wait their turn to be sent over a link that will get them closer to their destination. Subsequent packets are unlikely to follow the same path, so that the times they take to reach their destinations will vary and they may arrive out of sequence. The operation is simplex; any response is handled independently of the message received.

See Figure 1 below. You can see with connection-oriented protocol that addresses are used to establish the circuit prior to information exchange. It is torn down when exchange is completed. During exchange, links are used by this message only. Its operation can be simplex, half-duplex, or duplex. With connectionless protocol, the address is carried by message. At each node of the network, it waits its turn in queue and is directed towards destination. Subsequent messages are not likely to take same path through network. It uses simplex operation.

Figure 1
Connection-oriented and Connectionless
network connections



3. What is the TimeQuantum constant used for?

The time quantum constant is referred to in context of the round-robin scheduling algorithm, which is used for time-sharing systems. The time quantum is a small unit of time, usually from 10 to 100 milliseconds. In a time-sharing system, there may be many processes sent to the CPU at or about the same time. The time quantum represents the amount of time that each process has at the CPU for processing before the process is preempted and put back in the ready queue. Consider an example where the time quantum is equal to 20 milliseconds: If there are 10 process in the ready queue and each process is allowed 20 milliseconds (i.e., one "time quantum") of CPU time, a specific process will get 20 milliseconds of processing time with the CPU before the next process is queued and gets its own time quantum of CPU time. As different processes require different lengths of time with the CPU to complete, some processes will finish before others. Nevertheless, each one continues in round-robin fashion and has the CPU for at most one time quantum at a time until it completes processing. Each process gets as many cumulative time quantums as it needs to finish processing, but no more than one time quantum at a time before the next process gets another chance using one time quantum.

4. Why does the operating system need a stack?

An "interrupt" is a well-known and understood type of resource. Interrupts are used to demand attention from the central processing unit (CPU). This allows a device or subsystem to work in the background until a particular even occurs that requires system processing. After an interrupt is triggered, an interrupt handling routine saves the current CPU register states to a small area of memory (called the stack) then directs the CPU to the interrupt vector table. The interrupt vector table is a list of program locations that correspond to each interrupt. When an interrupt occurs, the CPU will jump to the interrupt

handler routine at the location specified in the interrupt vector table and execute the routine. In most instances, the interrupt handler is a device driver associated with the board generating the interrupt. For example, and interrupt request (IRQ) from a network card will likely call a network device driver to operate the card. For a hard disk controller, an IRQ calls the BIOS ROM code that operates the drive. When the handling routine is finished, the CPU's original register contents are "popped" from the stack, and the CPU picks up from where it left off without interruption.

Essentially, the stack allows memory to handle first-in/first-out (FIFO) demands in an orderly fashion. In effect, any routines are "stacked" on top of each other as they are received. An operating system actually has a lot of stacks. Another way of understanding a stack is this: The stack is used to store local data types and function pointers during the execution of a program. When you make a function call, all of the parameters passed into that function are stored onto the stack, and all local parameters declared within that function are stored on the stack. When the function returns, then all of the parameters stored on the stack for that function call are "popped" off the stack. Depending on the O/S, an operating system actually has at least one stack for each thread of execution.

A "stack fault" can be caused by several conditions, such as (a) defective CPU or memory, (b) improper or incorrect file access rights, (c) insufficient available memory {i.e., RAM}, or (d) an old version of a program running under Win95/98. It is important that programmers set the stacks properly. If you are running DOS only, you can usually remove the interrupt stack references entirely, or set it to 0,0. For Windows 95/98 systems, the setting of 9256 is adequate for most settings. To troubleshoot for a stack fault, the programmer should consider the following: (a) recheck System Configuration, (b) check available DOS memory, (c) check for STINV= environment variable (type SET at DOS), (d) check for Virus contamination, (e) check file access privilege (NOT Read Only), and (f) check for an update to the program.

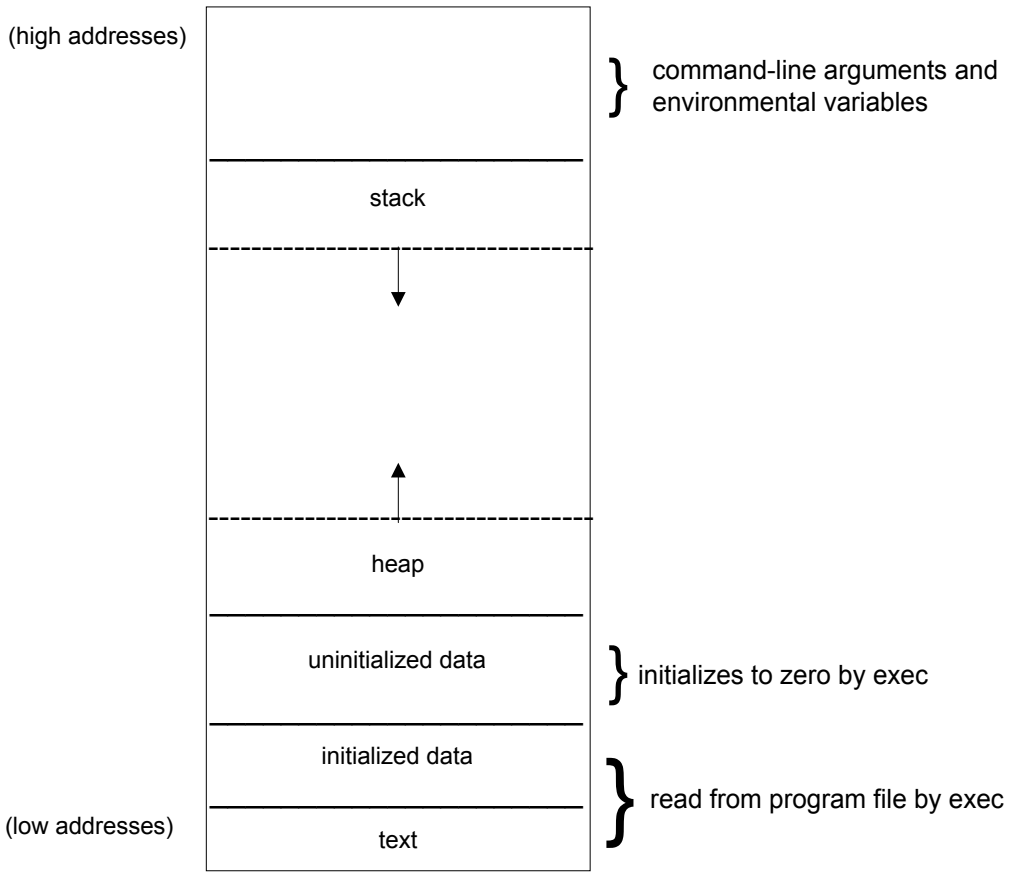
Programmers know that there are two ways to construct a stack from scratch, using either of the following commands:

Stack([initial_size]) ---returns a new empty stack instance allocating at least the given number of slots for stack elements. If the parameter is not given, a reasonable default is chosen.
StackFromSequence(seq) ---- constructs a stack instance from the given sequence. The instance is filled with all the elements found in the sequence by pushing the items from index 0 to len(seq)-1 in that order (i.e., popping all elements from the stack results in a reversed sequence).

Linux programmers consider the following executor command-line switches for allocating stack memory:
Stack n[k] ---uses "n" kilobytes of stack memory for the system
Stack n[mb] --- uses "n" megabytes of stack memory for the system.

Refer to Figure 2 below for diagram showing the typical memory arrangement for the UNIX system. With UNIX, the stack is where automatic variables are stored, along with information that is saved each time a functi0n is called. Each time a function is called, the address of where to return to, and certain information about the caller's environment (such as some of the machine registers) is saved on the stack. The newly called function then allocates room on the stack for its automatic and temporary variables.

Figure 2
UNIX: Typical Memory Management



5. Explain the difference between a process and a thread.

A process is a program being executed, and is defined by the resources it uses and by the location at which it is executing. A thread is like a mini process, or rather, like a number of processes that share resources and allow those resources to be shared at the same time. A thread is sometimes called a lightweight process because it shares resources with other threads (i.e., lightweight processes). A thread consists of a program counter, a register set, and stack space. What it shares with its peer threads is its code section, data section, and operating system resources such as open files and signals, collectively referred to as a task. A "process' is equal to a task with one thread. A task does nothing if no threads are in it. CPU switching among peer threads, and the creation of threads, is inexpensive when compared to the context switches among processes.

6. What are the disadvantages of using two operating systems in a multiprocessor system?

In any multiprogramming environment, several processes may compete for a finite number of resources. In a multiprocessor system, the processors share memory and a clock, and communication typically takes place through the shared memory. One of the problems of using two operating systems in a multiprocessor system is the issue of scheduling. Scheduling is more complex if multiple operating systems are in use than for single processor systems. If the processors are functionally homogeneous (i.e., identical), then any available processor can be used to run any process in the queue. If the processors are different (i.e., a heterogeneous system), then only programs compiles for a given processor's instruction set

can be run on that processor. There are sometimes limitations on scheduling even with a homogeneous multiprocessor. For instance, a system with an input/output (I/O) device attached to a private bus of one processor has the problem of processes needing to use that device must be scheduled to run on that processor; if they weren't, then the device would not be available for use.

Another thing to consider is that it can be a cumbersome to have an operating system support multiple file structures. With two operating systems in a multiprocessor system, the system can be even more cumbersome. The reason is that each operating system defines file structures and the more file structures that need to be defined, the more coding that is needed to support those file structures. Also, problems could result from new applications that need information structured in ways not supported by the operating system(s).

7. What is the basic idea of the multiple servers and clients IPC patterns?

Networks (and the client/server model) essentially are described as linkages between computers allowing data and other digitized information to be transmitted between computers. Cooperating processes can communicate in a shared-memory environment. One way is that these processes share a common buffer pool. Another way is for the operating system to give the means for cooperating processes to communicate with each other through an interprocess-communication (IPC) facility. IPC gives the ability to processes to communicate and synchronize their actions. IPC is best accomplished by a message system. The basic function of the message system is to allow processes to communicate with each other without the need to resort to shared variables. The IPC facility provides the send(message) and receive(message) operations. If two processes want to communicate, they must send messages to and receive messages from each other, and a communication link must exist between them. A client/server network is one of the ways the communication link can be set up.

8. What is the difference between a port and a message queue?

Processes that need to communicate with each other must have a manner in which to refer to each other. They can either use direct communication or indirect communication. With direct communication, each process that wants to communicate must specifically name the recipient or sender of the communication. With indirect communication, messages are sent to and received from ports (a.k.a., mailboxes). A port is tantamount to an object into which messages can be placed by processes and from which messages can be removed. Each port has a unique identification. The port has an initially empty queue of messages, but as messages are sent to the port, the messages are copied into the port.

Message queues are spoken of with regards to buffering. A port or link has a set capacity that determines the number of messages that can reside in it temporarily. This property can be viewed as a queue of messages attached to the link. There are three ways that a queue can be implemented: (a) zero capacity {a message system with no buffering}, (b) bounded capacity {the queue has a finite length}, and (c) unbounded capacity {the queue has potentially infinite length}.

Note: A port can be owned either by a process or by the operating system. If the port is owned by a process, then it is necessary to distinguish between the owner and the user of the port.

9. What are the three general strategies for adding a new facility to a system?

"Code" is written computer instructions, and can appear in a variety of forms. The code that a programmer writes is called source code. After it has been compiled, it is called object code. Code that is ready to run is called executable or machine code. To add new functionality to code, or to expand a system, consider the following four strategies:

(a) Recompile or modify the source code itself. Compiled programs run significantly faster than interpreted ones because the program interacts directly with the microprocessor and doesn't need to share memory space with the interpreter.
(b) Link different modules together for increased or expanded functionality.
(c) Use a table system to call different subroutines, which will provide expended functionality. A subroutine is a subsidiary routine within which initial execution never starts. It is executed when called by some other program, usually the main program. It is also called a subprogram. Essentially, a subroutine is a set of instructions, given a particular name that will be executed when the main program calls for it. There are two main advantages to using subroutines when writing programs. First, if a complicated set of instructions is needed at different locations in the

program, then making instructions into a subroutine can save considerable work since it will be unnecessary to type the instructions more than once. Secondly, a complicated program is best written as a collection of smaller parts, each with a well-understood purpose.

(d) A Layered approach can be taken with system design. Modularity is the main advantage of the layered approach. The layers are selected such that each uses functions or operations and services of only lower-level layers. This approach can simplify debugging and system verification.

10. Why are there two levels of memory management?

The two levels of memory management are represented by main memory and secondary storage (disk). The two main reasons that there are two levels of memory management are (a) secondary storage is much larger and accommodates more data and programs, and (b) main memory does not permanently hold data; secondary storage stores data. Executing programs is the primary purpose of a computer. The programs and the data they access must be in main memory during execution. In fact, the main memory is typically the only large storage device that the CPU is able to address and access directly. For the CPU to process data from disk, it must transfer it to main memory by CPU-generated input/output (I/O) calls. The reason that it is on disk (i.e., secondary storage) is that the main memory is too small to hold all data and programs. Secondary storage backs up data used in main memory and stores it. When the computer is turned off, data in main memory is lost.

11. Why is it called virtual memory?

Virtual (or logical) memory is a concept that, when implemented by a computer and its operating system, allows programmers to use a very large range of memory or storage addresses for stored data. The computing system maps the programmer's virtual addresses to real hardware storage addresses. Usually, the programmer is freed from having to be concerned about the availability of data storage.

In addition to managing the mapping of virtual storage addresses to real storage addresses, a computer implementing virtual memory or storage also manages storage swapping between active storage (RAM) and hard disk or other high volume storage devices.

Virtual memory is essentially a way of extending the effective size of a computer's memory by using a disk file to simulate additional memory space. Typically, the O/S keeps track of which memory addresses actually reside in memory and which memory addresses must be brought in from disk when they are referred to. Virtual memory is essentially a technique that allows the execution of processes that may not be completely in memory. The main advantage of virtual memory is that programs can be larger than the computer's physical memory. So, it essentially separates logical memory as seen by the user from the physical memory. It is called virtual memory because it is like an imaginary or simulated memory area supported by the operating system in conjunction with the hardware. Virtual memory is like an alternate set of memory addresses. Programs use these "virtual" addresses rather than real addresses to store instructions and data. Once the program is actually executed, the virtual addresses are converted into real memory addresses.

The management of virtual memory operations is at the operating level rather than the application level. A significant advantage to implementing virtual memory at the operating level rather than the application level is that any program can take advantage of the virtual memory, with the result that memory extends seamlessly from RAM to the computer's secondary storage.

12. What is a page fault? What is a typical page fault rate?

Virtual storage is divided into units called "pages," and the operation of bringing in a page from a disk is called a "page fault." A page fault is the situation that arises when the computer needs to access an area of memory that has been swapped out to disk. When a program needs a page that is not in main memory, the operating system copies the required page into memory and copies another page back to the disk. Each time a page is needed that is not currently in memory, a page fault occurs. An average page fault rate is 25 milliseconds.

13. Give some reason(s) to lock a page in memory.

Sometimes, pages need to be locked in memory when demand paging is used. One occurrence of this would be when I/O is done to or from virtual memory. Locking a page in memory keeps a process in memory instead of it being replaced by a global replacement algorithm. When pages are locked into

memory, a lock bit is associated with every frame. When the frame is locked, it can't be selected for replacement by the global replacement algorithm. Once the I/O is complete, the pages are unlocked.

14. What is a theory of program behavior? Give two examples.

A program is a "passive" entity, such as the contents of a file that is stored on disk. In contrast, a process is an "active" entity. As an active entity, a process has a program counter that specifies the next instruction to execute. Two processes may be associated with the same program, but are considered two separate execution sequences. "Program behavior" implies a process. That is, it implies an activity or execution of something. A program such as Microsoft WORD is saved on the disk, and is in latent (passive) state until needed. When being used different processes happen "actively" in WORD, such as the font type being changed for a sentence, or if new text is added to a certain section of a report.

A couple of examples of how the operating system is responsible for activities in connection with the management of processes are: (a) the provision of mechanisms for deadlock handling, and (b) the creation and deletion of both user and system processes.

In contrast to application programs, system programs provide a convenient environment for program development and execution. A couple of types of system programs are (a) those for program loading and execution, and (b) those for file modification.

15. Why are variable sized objects hard to deal with in a program?

Computer systems are collections of processes and objects. Objects can be both hardware objects (e.g., CPU, memory segments, printers, disks, and drives), and software objects (e.g., files, and programs). A certain amount of memory is set aside for an object when it is created. If the size of the object is changed, you need to expand the memory space assigned to that object. Any instructions that are put in for an object follow one step after another, one at a time. If instructions were figuratively illustrated as "1,2,3" for a large chunk of the program, and then comes along a variable instruction "1,2,3,4,5,6," you can see (in this context) that it needs more attention; it needs more handling to get the job done.

The difficulty of programs being able to deal with variable-sized objects may depend somewhat on the programming language. In the C programming language, it is harder because you have to keep track of the memory yourself (in the programming language), forcing you to keep a pointer and the length of memory in program logic. It is fairly easy to lose track of these in the program (by overwriting the variable or forgetting to free the memory). Since variable-length memory is taken from the heap, it will remain throughout the execution of the program, so if you forget to free it, or lose track of the pointer, then it will become wasted memory (a.k.a., memory leak). Fixed-sized memory is often (though not always) allocated from the stack, so when your procedure returns it is popped off the stack and automatically freed (cannot be leaked).

16. Contrast seek time and latency time.

Seek time is used in reference to a magnetic disk and is the time it takes to move the disk arm to the desired cylinder. It is part of what is referred to as "random access time" (the other part being the time for the desired sector to rotate to the disk head). Latency time is one of the two aspects of the speed of tertiary storage (the other aspect being bandwidth). Latency time is a performance measure and disks are much faster than tapes. So, seek time is the time it takes the disk arm to go to the specific cylinder on a magnetic disk drive. Latency time is the time it takes to gain access to data, whether on a magnetic disk drive, or a tape drive. A typical hard disk has an average latency time of 8 milliseconds, and an average seek time of 15 milliseconds.

17. Why do SCSI device requests often have to go through two device drivers?

SCSI stands for **S**mall **C**omputer **S**ystem **I**nterface, and it is the controller board that remains the preferred hardware to add to hypermedia computers. The main use is to connect the computer to auxiliary storage drives such as CD-ROM drives, auxiliary hard drives, and computer tape drives.

A general-purpose computer system consists of a CPU and a number of device controllers that are connected. Each device controller controls a specific type of device. With a SCSI controller, there can be several devices attached to it at once. The SCSI controller maintains some local buffer storage and a set of special-purpose registers. It is responsible for moving the data between the peripheral devices that it

controls and its local buffer storage. Because there may be more than one device attached to the SCSI, device requests may have to go through two device drivers before they are implemented.

18. Why are directories implemented as files?

Directories are implemented as files to improve the efficiency, performance and reliability of the file system. A directory contains a set of files or subdirectories, and is itself simply another file. However, it is treated a little differently. All directories have the same internal format in that one bit in each directory defines the entry as a file (0) or as a subdirectory (1).

File systems on computers can be extensive. To manage all the data, we need to organize them. Each disk on a computer system contains at least one partition where files and directories are kept. Each partition contains information about files within it. This information is kept in entries in a directory, and the directory records information for all files on a partition such as size, type, file name, and location. The directory is like s symbol table that translates file names into their directory entries. Whenever a file name is given to be loaded, the operating system (O/S) searches the local user directory for the file.

To create a new file, an application program calls the logical file system, which, in turn, reads the appropriate directory into memory, updates it as necessary with the new entry, and writes it back to the disk. Some operating systems (e.g., UNIX) treat a directory exactly as a file—one with a type field indicating that it is a directory.

Whenever a file is read, a field in the directory structure needs to be written to. Any time a file is open for reading, its directory entry must be read and written as well. The easiest way to implement a directory is to use a linear list of file names with pointer to the data blocks.

19. What is a digital signature?

A digital signature is an electronic (rather than a written) signature that can be used by someone to authenticate the identity of the sender of a message or of the signer of a document. It can also be used to ensure that the original content of the message or document that has been conveyed is unchanged.

Additional benefits of the use of a digital signature are that it is easily transportable, cannot be easily repudiated, cannot be imitated by someone else, and can be automatically time-stamped.

A digital signature can be used with any kind of message, whether it is encrypted or not, so that the receiver can be sure of the sender's identity and that the message arrived intact. A digital certificate contains the digital signature of the certificate-issuing authority so that anyone can verify that the certificate is real. Here is how it works: Assume you were going to send a document to your accountant in another city. You want to give your accountant the assurance that it was unchanged from what was sent by you and that it really is from you. You copy-and-paste the document into an e-mail note. Using special software, you obtain a message hashing (mathematical summary) of the contract. You then use a private key that you have previously obtained from a public-private key authority to encrypt the hash. The encrypted hash becomes your digital signature of the message. (Note: it will be different each time you send a message.) At the receiving end, your accountant receives the message via e-mail. To make sure it's intact and from you, your accountant makes a hash of the received message. Your accountant then uses your public key to decrypt the message hash or summary. If the hashes match, the received message is valid.

20. What is a FAT file structure?

FAT stands for “file allocation table.” It is a simple and efficient method of disk-space allocation and is used by the OS/2 and MS-DOS operating systems. FAT is a variation on the linked allocation method. With linked allocation, each file is a linked list of disk blocks, and the disk blocks may be scattered anywhere on the disk. Each block contains a pointer to the next block. A section of disk at the beginning of each partition is set aside to contain the table. The table is indexed by block number. The FAT is used like a linked list. The directory entry has the block number of the first block of the file. The table entry that is indexed by that block number then has the block number of the next block in the file. An advantage of this system is that random access time is improved, because the disk head can find the location of any block by reading the information in the FAT. This system also helps prevent external fragmentation of stored data. With FAT, linked allocation can support efficient direct access to blocks of data because the linked list is placed in one contiguous area, thereby resulting in (a) improved use of disk space, (b) improved performance of the file system, and (c) improved reliability of secondary storage. So,

to summarize, FAT is a file system structure used by MS-DOS and OS/2 for organizing and tracking file storage, and it is the part of the disk that contains information about the sizes and locations of files.