Computers & The Cloud Answers

1. A system is a set of things working together as parts of an interconnecting whole. It’s a complex whole.
2. Abstraction is the removal of excessive, redundant and unnecessary details from a system, task, visualization or algorithm, in order to focus on the essence / important parts of the system.
3. Abstraction can be used to separate a system into (abstraction) layers. Layers rely on the layer directly below them. The details of lower layers are hidden to focus on the task at hand. This makes problems more manageable and easier to solve. There is decoupling between layers, meaning lower layers are independent of higher layers – which could make the system more secure. Each layer has a specific responsibility. Encapsulation – process of adding headers and trailers around data,
4. The command line is used to execute commands. It’s an interactive way of programming. It has many special purpose tools. It’s more complicated, but easier to adapt and combine.
5. GUIs are made for a specific purpose. They are often fixed and have limited adaptability and combinability.
6. A protocol is a set of rules governing the exchange or transmission of data between devices. They specify the sequence of messages, and format of data in the messages. They allow computers to communicate regardless of their manufacturer.
7. There are many different types of devices from many different manufacturers on the internet. It’s impossible to make a custom version of an application for each one, so protocols define a common set of rules for data transmission and exchange so all devices can use the application.
8. Protocols can stack to form protocol stacks.
9. UDP, TCP, IP, XMPP
10. Extensible messaging and presence protocol defines how to exchange messages between client and server.
11. XMPP is classed as an application-level protocol because it addresses users instead of servers.
12. Internet protocol defines how to exchange data packets between internet addresses.
13. IP is classed as a network level protocol as it addresses servers instead of users.
14. Hardware (Provides computing and storage capacities) → Virtual Machine (Share computing/storage between many tenants) → Operating System (manage hardware, provide drivers, scheduling, file systems) → Containers (isolate apps, their libraries and dependencies → Application (provides desired functionality to user)
15. Hardware is very low level (many details), hardware varies from one computer to another (heterogenous)
16. The principles of the von neumann architecture are: programs are stored in memory with data, I/O interfaces, CPU, memory, Buses carry data around.
17. Fetch decode execute cycle.
18. Compilers are programs that translate high level languages to lower level languages
19. The operating system is an abstraction layer of software that sits between hardware and other software. Group of programs that act as a bridge between the user and the computers hardware and manage the operations of the computer for the user. Modern computers are very complicated systems, often containing many cores, and connected to a multitude of peripherals. This forms part of the reason software developers don’t think in terms of hardware. The variety of components are hard to track and vary between manufacturers. The OS manages the components of a computer and ensures they operate at a high-performance level, which can be hard to do without using an OS.
20. The OS runs in kernel mode
21. Kernel mode refers to when software has access to all computer hardware and can execute any instruction the machine is capable of executing.
22. Other software runs in user mode. Which has a limited number of machine code instructions available
23. Shell is a computer program that exposes an operating system’s services to a human user or group of programs. It’s commonly a command line interface
24. Portable operating system interface x (for unix)
25. A screenshot of a computer screen

    Description automatically generated with medium confidence

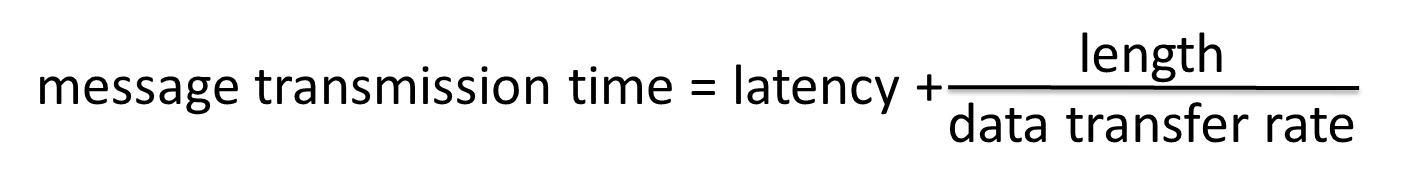
1. Makes applications portable across many devices – interoperability
2. System calls are function calls at the boundary between user mode and kernel mode. User programs make system calls, which invoke the operating system and call its code. POSIX lists system calls, and some of the standard c library.

|  |
| --- |
| Pros |
| OS turns the CPU to a high-level interface which: |
| Enables multiple programs (processes) to run in parallel |
| Each process can only access well-defined area of memory (memory isolation) |
| Each program can have many threads that seem to run in parallel |

1. Hard drive – OS implements file system with tree like structure. Primary memory - 2 storage areas for global variables:
   1. Stack – local variables, stack frame made / pushed when function called
   2. Heap – dynamically allocated memory for global variables which can become fragmented
2. A process is a program in execution. Unit of activity characterized by single sequential thread of execution, a current state, and an associated set of system resources
3. Executable program, execution content of the program, data needed by the program
4. A thread is an independent unit of computation that can be scheduled for execution on a CPU. Threads are created within a process and can share process address space (edit the same variable in a process). Single threading – one thread per process. Multi-threading - multiple threads per process
5. Enhance parallel processing, increase response to the user, utilize idle time of the CPU (example on slides 20), prioritize work (scheduling?)

|  |  |
| --- | --- |
| Process | Thread |
| More time to create and terminate than thread | Less time to create and terminate than process |
| Easier to do concurrent programming | Easier to do parallel processing |
|  | Less time to switch between 2 threads in the same process than between 2 processes |
|  | Threads can communicate via shared memory |
|  | Using shared memory is risky |

1. Middleware is a layer of software that sits between the OS and apps that is needed to help other software to run (e.g., java virtual machine)
2. A packet is a logical unit of transmission that is part of a longer message. It has a maximum length and source/destination addresses.
3. Packets are transmitted using packet switching. Packets are transmitted by a store-and-forward network. The packets are first stored in memory before the node in the network decides which link bring the packet closer to its destination (process of routing) and forwards the packet.
4. Packets may travel along different routes, packets may get lost, packets may get corrupted, packets may arrive out of order.
5. Hop limit – limit existence of packet in a network. Reduced at every hop.
6. Open systems interconnection protocol model – conceptual model where each layer extends the properties of the underlying communication system.
7. There is encapsulation between layers. Lower layers independent of higher ones. Protocol data units passed between layers
   1. Application – human computer interaction layer where app can access network services - differs depending on service offered. End user layer. Enables network apps to communicate
   2. Presentation – ensures data is in a useable format – where encryption occurs – defines data format
   3. Session – maintains connection and controls ports and sessions -Defines how to establish a session between systems
   4. Transport – enables multiple apps on same host, provides services like transmission – TCP, UDP. Establish connection between computers. Flow + error control. Breaks data intro chunks called segments
   5. Network – Blocks of info routed across a network of data links IP address – IP. Breaks segments into packets. Facilitates different network transfer
   6. Data Link – Routing managed. MAC address. Encapsulates a packet in a frame. Frame header – source/destination mac address. Trailer – frame check sequence field – error checking data. Facilitate ssame network transfer.
   7. Physical – Bits sent across transmission medium. Defines how to move bits between devices. Includes physical equipment, DPackets – bit stream. Agree on signal convention
8. On sending, message goes down layers, on receiving it goes up the stack
9. Internet protocol enables communication between networks. Delivers packets.
10. IPV4 and IPV6.
    1. IPV4 – address space too small for modern apps (not enough unique addresses), 32 bit addresses, decimal, 4 8bit val
    2. IPV6 – 128 bit addresses, hex grouped in 8 16 bit val
11. Header – ip version, length, service code (priority), hop limit, source/destination address
12. 1. TCP – reliable end-to-end transmission , establishes connection before transmitting payload, data streams. Ordering of packets due to connections. Bidirectional
    2. User datagram protocol – best effort (successful transmission not guaranteed), single packets. Connectionless data transmission. More efficient and faster than TCP. Unreliable
13. Port number identifies a service/application on a device
14. HTTP uses port 80. HTTPS uses port 443
15. Three-way handshake ensures both ends of the connection are ready to send and receive data. ISNs keep track of data segments exchanged
    1. Step one: syn (synchronize) – sender sends receiver a tcp packet with a random sequence number
    2. Step two: syn + ack (synchronize and acknowledge) – receiver responds with tcp syn and ack packets. Received sequence number is increased by one and sent back for acknowledgement. Recipient also generates own random initial sequence number
    3. Step three: Handshake complete – Sender increments the initial sequence number of the recipient. Senders syn is included in ack field of packets
16. Latency is the delay that occurs after a send operation is executed



1. HTTP is a request response protocol. It’s used to access resources. Resources are addressed with uniform resource identifiers
2. OS aren’t designed to isolate resources for many users + programs depend on their environment and have dependencies. VMs and containers provide isolation
3. Containers are lightweight packages of applications along with their dependencies (e.g. libraries) which are required to run the app.
4. A container engine runs as a normal app on top of the OS. Each container above this runs as a normal OS process, enjoying memory isolation. Containers above the same container engine run the same OS. Containers share the OS kernel of the host OS.
5. Virtual machine manager runs on top of the underlying hardware. They are responsible for running and creating virtual machines. Each vm above the hypervisor thinks they run on their own hardware, so they each need their own operating system. Each vm can have a different os than the host os, and they don’t share the os kernel with the host os.
6. 1. Fidelity – software on the hypervisor executed identically to execution on hardware
   2. Performance – Majority of the guest instructions are executed by the hardware without the intervention of the VMM
   3. Safety – VMM manages all hardware resources
7. 1. Native (bare metal – better performance – hypervisor operates directly above hardware system
   2. Hosted – More practical – hypervisor operates directly above OS

|  |  |
| --- | --- |
| Container | Virtual machine |
| Memory and resource isolation | Memory and resource isolation |
| More lightweight than virtual machines | Less lightweight than containers |
| Can’t have different OS to host OS | Can have different OS than host OS |
| More portable than VMs |  |
| Makes development process more modular |  |
| Uses less memory than VMS |  |
| Exploit in one container could affect shared hardware with other containers |  |

1. Docker is a container technology
2. List of instructions on how to create an image
3. Cloud computing is the remote provision of it resources remotely over the internet to store, manage, and process data, rather than doing it using a local server or personal computer.
4. SAAS PAAS IAAS FAAS
5. 1. Software as a service – everything is managed by provider. Product run and managed by the service provider. Don’t worry about how the service is maintained. It just works and remains available. No maintenance by us. No software development by us. Cost efficiency (no capex). Rapid deployment. Continuous updates. No control over app (provider outage, features). Need internet to access. Security concerns. Limited offline functionality.
   2. Platform as a service – Hardware and system software managed by CSP. OS, container system, runtime environment for python, middleware , physical hardware, physical security, network infrastructure, network security. Dw about understanding hardware or OS or other system software. Cost saving, streamlined production, reduction of on premises opex. Cost efficient (lower capex), reduced complexity. Might not be much saving with very high capex. Vendor lock in / dependency. Reduced control over infrastructure. Security concerns – sharing.
   3. Infrastructure as a service – Hardware managed by cloud service provider. Includes: physical hardware maintenance, physical security, network infrastructure, network security. Provides access to networking features, computers and data storage space. Dw about it staff, data centres, hardware. Still mainr Very flexible, More control over development environment. More complex than Paas, security concerns
   4. Function as a service – paas + predetermined architecture and app life cycle management. Example is aws lambda. Allows you yo execute code in response to events without complex infrastructure of microservice apps. Decouples parts of system and avoids changes to main app. Can use separate code bases. Cold start latency – when a function is invoked for the first time or after a period of inactivity it may experience a cold start, which introduces additional latency. Vendor lock in
6. 1. Public cloud – everything built on the CSP. Computing resources shared with other customers
   2. Private cloud – Everything built on companys datacentres. High upfront costs/ capex. Some efficiences of cloud computing whilst taking responsibility for your data. Pooling and sharing of data and resources across apps and departments.
   3. Hybrid cloud – Use of both public and private cloud. public cloud for general computing. Saves some upfront cost. Good when full migration is tough.
   4. Community cloud – private cloud shared by multiple companies, often with shared interests or compliance concerns.
7. General data protection regulation
8. Eu citizens
9. Data must be accurate and maintained accurate. Right to have data erased. Right to restrict processing. Only necessary data.
10. Confidentiality integrity availability
11. The environmental impacts of it. Could increase ability to work from home. Privacy. GDPR. Specialisation. Innovation.
12. A distributed system is a collection of autonomous computers that are connected through a network and middleware. Users of a distributed system perceive the system as a single integrated computing facility.
13. Middleware is a distribution software that enables computers to coordinate their activities and share resources of the system.
14. Data centres
15. 1. Some data centres have specialised components.
    2. Switches, ethernet and fiber optic cables used for connectivity.
    3. 3-tiered design – core – connectivity to many aggregation switches, manage traffic aggregation – load balancing + extra stuff access – servers in racks connect to network
    4. Uninterruptible power supply, redundancy in power supplies
16. Specialised version of mainstream OS is used, Red hat linux.
17. Offer a single system view. Addresses distribution challenges like: distributed file system, messaging middleware, database middleware. Middleware layer hides the complexity of the distributed system from the users. Solves different things depending on deployment context.
18. Partial failures, parallelism, communication through message passing, independent clocks
19. 1. Heterogenity of components (os, hardware, programming language)
    2. Openness – can system be re-implemented in different ways (no vendor lockin for example)
    3. Security
    4. Scalability – will system remain effective with #users increasing
    5. Failure handling – redundancy, disaster recovery
    6. Concurrency – multiple people access simultaneously
    7. Transparency/abstraction – how to hide complexity of system
    8. Quality of service
20. 1. Centralised architecture (client-server) – server implements a service; client requests a service from a server.
    2. Decentralised architecture (peer-to-peer) – Each node operates its own share of the data set. Responsibilities less centralised
    3. Hybrid architecture – combination of client-server and decentralized architectures
21. Uncertainty of communication - can never be totally sure of successful communication. Mitigate the communication.
22. 1. What are the entities
    2. How do they communicate
    3. What are their roles
    4. How are they mapped to the physical architecture?
23. Nodes have identifiers. Links connect nodes. Synchronous communication. Limited communication bandwidth (congest name). Global clock
24. 1. Leader election – computers track a leader variable. When all nodes have same leader variables and stop changing leader variable the leader is set
    2. Single source shortest path – given a node, we want any node to know how many links to that node and which neighbour is on the way to the node
    3. All pairs shortest path – Nodes should know how many links there are to another node and which neighbour is on the shortest path to that node
25. Each node has bool variables done and leader. Leader set for one node. Leader sends wave to all neighbours. For non-leaders – if done do nothing else wave to all neighbours
26. Breadth first search algorithm. Leader given. Each node has: distance to leader, node parent which is on shortest path to leader, children nodes, counter done\_children and state (wait\_proposal, send\_proposal, wait\_accept2, wait\_accept1, wait\_ack, done. Distance undefined for nonleaders, 0 for leader. State wait\_proposal for nonleaders and send\_proposal for leader.
27. 1. Elect a leader
    2. Leader circulates token in depth first search order with speed 0.5 (token moves every 2 steps)
    3. When node first sees token, it initiates BFS with itself as leader
28. 1. Each node orders its neighbours in a circular queue
    2. If token received for first time or from neighbour it was last sent to, send it to next neighbour in circular queue
    3. Else token received from other neighbour then send it back to that neighbour
    4. When leader gets back token from all neighbours (as next neighbour) stop
29. Byzantine and non-byzantine failures
    1. Non-byzantine failure – regarded as crash failures. A component or process in the system stops functioning or crashes but doesn’t show any malicious behaviour. Node stops working at any time, may start working again at any time. Can’t tell from lost messages. Can be detected and mitigated using fault tolerance techniques (redundancy, error detection)
    2. Byzantine failure – Component or process stops following normal protocol and starts acting randomly. Nodes may act adversarial and show malicious behaviour, by sending incorrect or misleading information to other nodes in the system. Much harder to detect and mitigate. Caused by many factors
30. Hard to reason about byzantine failures
31. Acceptors and proposers. Used for consensus on next request to handle. Proposers can send prepare (proposal id) and the acceptor replies with promise(proposalid, previous accepted proposalid, previous accepted value) or nopromise (proposalid) (if new proposalid > = old one. For propose(p,w). if p> old id then its accepted.
32. File
33. The file system is the component of the operating system that manages files. They provide an abstraction layer above raw memory.
34. Logical unit of information
35. Name, content, type, attributes
36. Contiguous allocation – store file as sequence of disk blocks , only remember start address. Disk defragmentation over time. Use defragmentation software. Space saved due to no pointers.

Alternatively use a linked list, with a next variable pointing to where the file storage continues. Every disk block used. No space lost. Fast sequential file reading. Random access. Some space lost due to pointer storage

1. Many programs append data to the same file without conflicts
2. 1. Strict – any update must be seen immediately by all other entities
   2. Eventual – if no update takes a long time, all programs will converge on the same result
3. Files divided into fixed size chunks. TCP and UDP used, chunkservers storing same chunks should be on different racks. Master, replicated master, chunkserver, client apps
4. The purpose of distributing computation is performance and saved time, but the correct result must still be produced.
5. Order programs are executed has no effect on overall result. Programs can’t depend on each other. Dependency – one program reads from a variable and another program writes to that variable, vice versa, or both programs write to the same variable
6. Different iterations of a loop depend on each other