**Algorithmic Decomposition** — Top-Down structured design. — Modules are the building blocks. — **Functional Decomposition**: operations. — Routines represent execution steps in some overall process. **Object-Oriented Decomposition** — Class Definitions: Couple data with their operations. — Object instantiation from class definitions. — Object interacts through messages. — Objects have their own state that change throughout their life time. **Object Oriented Analysis (OOA)** A method of analysis that examines the target system's requirements from the perspective of classes and objects. ● **Object Oriented Design (OOD)** A method of design adopting object decomposition, and a notation depicting all types of models with the target system domain. ● **Object Oriented Programming (OOP)** Implementation method organizing cooperative collection of objects, each of which is an instance of some class. (data)intialized global and static variables. (bss)uinitialized global &static variable. **A definition entails implicit or explicit declaration** ● inline is a keyword that can precede a function definition and causes the compiler to inline the body of a function call to eliminate overhead**. Unlike macros, inlines take place in the compilation phase and not the preprocessing phase** **OO Model:** Major Elements: Abstraction, Encapsulation, Modularity&Hierachy. Minor Elements: Concurrency, Typing, Persistence . **Abstraction**: One way that humans use to cope with complexity. ●Arises from the recognition of similarities. ●Simplified description of specifications. ●A concept qualifies as an abstraction if it can be described, understood, and analyzed. ●Emphasis important details and withdraw unimportant ones. Denote the essential characteristics of an object that distinguish it from all other kinds of objects. ●Crispy defined conceptual boundaries. ●Focuses on the outside view of an object, and used to separate objects. ●Can capture the entire behavior of an object avoiding surprises and side effects. **Abstraction types:** ● Entity: An object that represents a useful model of a problem domain or domain entity solution. ● Action: Provides generalized set of operations. ● Virtual Machine: Group operations that are all used by some superior level of control. ● Coincidental: Package a set of operations that have no relation to each other. **Encapsulation:** Secret implementation of abstraction. ●Abstraction and encapsulation are complementary concepts: *abstraction focuses on observable behavior while encapsulation focuses on implementation that leads to this behavior*. ●Provides explicit barriers among different abstractions. ●For abstraction to work implementation must be encapsulated. Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation **Pure Virtual:** A pure virtual method has no implementation. ● All subclasses must declare and implement pure virtual methods of their base classes. **Final specifier:** used to disable overwriting and extending methods of a base class by its descendants. ● It allows and grant control to the base class to secure some of its method functionalities and make it persist through out the inheritance tree.**Important: non of the inheritance types, public, protected, or private, can give a descendant class access to the private data members and methods of the parent class.** **Classes assumes data members with no access modifiers as private by default while structs assumes them to be public Volatile disables code optimization. Diamond problem**, do virtual inheritance for the intermediate classes, there is a solution. **●Arrays sizes are defined at compile time. ●Vectors does not have predefined size and get expanded as needed at run time**. **Unlike macros, inlines take place in the compilation phase and not the preprocessing phase.** **Network:** A communication medium for data exchange between computing entities. ● A typical network is built up of: —Physical Layer: based on electronics. —Communication Protocols: Synchronization, Routing, Data presentation, and Service Delivery Interfaces. —Communication APIs: A mean for applications to use the communication protocols. **Packet Frame:** A stream of data built up of bits. ● The minimum unit of data communication between two computing entities. ● Packet frame size is configurable. ● Typically build up of a Header and a Payload. ● Nested Headers: Payload can act as a subpacket frame which contains a sub-header and a sub-payload. **Network Stack**: A modular perspective of interpreting packets within a computing entity. ● Sending Data: —Traverse stack top-down. —Construct corresponding stack level headers. —Forward Packet Frame to the next lower stack level. ● Receiving Data: —Traverse stack bottom-up. —Consume/Process stack level headers. —Forward payload to the next upper stack level. **ARP Protocol:** Address Resolution Protocol. ● Map IP addresses to Hardware Physical addresses. ● Operates in the network layer. ● Composed mainly of Request/Reply ARPs. ● A client send an ARP broadcast request asking about who has a specific IP address. ● The network node having the IP responds with ARP reply. ● The ARP reply is cached by the ARP clients for future use to reduce the ARP traffic on the network. ● ARP Proxy allow a node to respond to ARP requests on behalf of other nodes. **TCP:** Reliable and stateful; requires connection establishment. ● Provides a virtualization of communication session between 2 processes. ● Flow control capabilities through TCP windows. ● Packet ordering; can entail buffering on the implementation level. ● Packet reception acknowledgment. ● Entails some overhead. **UDP:** Unreliable connection-less data communication. ● Introduces process-to-process communication rather than host-to-host. ● Processes are mapped to service port numbers. ● Checksum capabilities for the whole packet. ● No packet reception guarantees; packets can drop. ● Out of order packet reception. ● No flow or congestion control. **UDP vs TCP**: UDP requires that the programmer to handle reception confirmation, flow control, and packet reorder. ● TCP layers the programmer from all the details and provides a file stream perspective of the underlying communication, with some overhead penalty. ● It is up to the programmer to choose between TCP and UDP based on the target application. **Sockets** ●A Virtualization Perspective of a communication medium. ●Virtual Channels based on Internet addresses (IP) and port numbers. ●A perspective for processes communication rather than devices communication. ●Provides a set of APIs for application to communicate over UDP or TCP. **Pthreads:** ● POSIX threads is an independent parallel execution model from a specific programming language. ● Designed to allow developers to control multiple parallel flows of execution that can overlap. ● Multiple implementations do exist that follows the POSIX thread specifications. ● Available on many POSIX compliant Unix-Like Operating Systems. **Pthread\_create:** Creates a new thread of control that starts execution by invoking the function start\_routine. The “void \* arg” is passed to the start\_routine function. The thread is created with the attributes defined in “pthread\_attr\_t \* attr”, and the default attributes are applied if “attr” is NULL. On success a 0 is returned, else an error number is returned. **Critical pthread**: A code section that needs to be run by one thread at a time. ● Need to be guaranteed that no two threads will be in the critical section simultaneously. ● No assumptions may be made about the speed to the concurrent threads racing for a critical section. ● No process should have to wait forever (Deadlock) ● ***Need an atomic locking mechanism to achieve critical section management.*** **Bounded Buffer Problem:** A shared buffer by more than one concurrent thread. ● Acting like a FIFO queue. ● Two types of threads: —Producers: Produce items from buffer. —Consumers: Consume items from buffer. ● Synchronization mechanisms are needed to manage the buffer. ● Mutual exclusion should be applied when changing the queue pointers and adding or removing items from the queue buffer. **Thread c++11:** C++11 has native thread support. ● Part of the standard library. ● Provide Object Oriented threads abstraction. ● Realize object lifetime and exceptions ● Provides several classes and templates for mutexes, locks, guards, and condition variables. ● Can integrate with new C++11 features like lambda functions. ● Lacks some of the features provided by pthreads such as pthread\_cancel and multi-core affinity features. **Static Library:** Statically linked within the executable and the object code is added to the final executable file, and the library must be present at compiler time. **● Shared Object:** Libraries that are used/shared by multiple programs and usually is provided system wide. **The library must be present at compile time, and when the application starts. e.g. -lpthread** **● Dynamic Shared Object:** The library is linked/loaded with programs at run time through a set of DSO APIs. The loading program should have some knowledge about the target DSO to be able to load it and execute function from it. **The library does not need to be present at program compile and startup time. Application Layer Protocol:** ● Highest level in the OSI (Open Systems Interconnection) model. ● Interpreted/processed in the user mode. ● Concerned with process-to-process, or even thread-thread, communication. ● Depends on the underlying transport protocol layer for data transfer. ● Abstracts interface methods and APIs between communicating entities. ● Examples: FTP, Telnet, NTP, DNS, SMTP, POP, SSH, TLS/SSL, and HTTP/HTTPS. **HTTP:** Simple in nature; every web transaction is based on a Request/Reply model.