HIERARCHICAL ARCHITECTURE

Hierarchical architecture views the whole system as a hierarchy structure, in which the software system is decomposed into logical modules or subsystems at different levels in the hierarchy. This approach is typically used in designing system software such as network protocols and operating systems.

In system software hierarchy design, a low-level subsystem gives services to its adjacent upper level subsystems, which invoke the methods in the lower level. The lower layer provides more specific functionality such as I/O services, transaction, scheduling, security services, etc. The middle layer provides more domain dependent functions such as business logic and core processing services. And, the upper layer provides more abstract functionality in the form of user interface such as GUIs, shell programming facilities, etc.

It is also used in organization of the class libraries such as .NET class library in namespace hierarchy. All the design types can implement this hierarchical architecture and often combine with other architecture styles.

Hierarchical architectural styles is divided as −

- Main-subroutine
- Master-slave
- Virtual machine

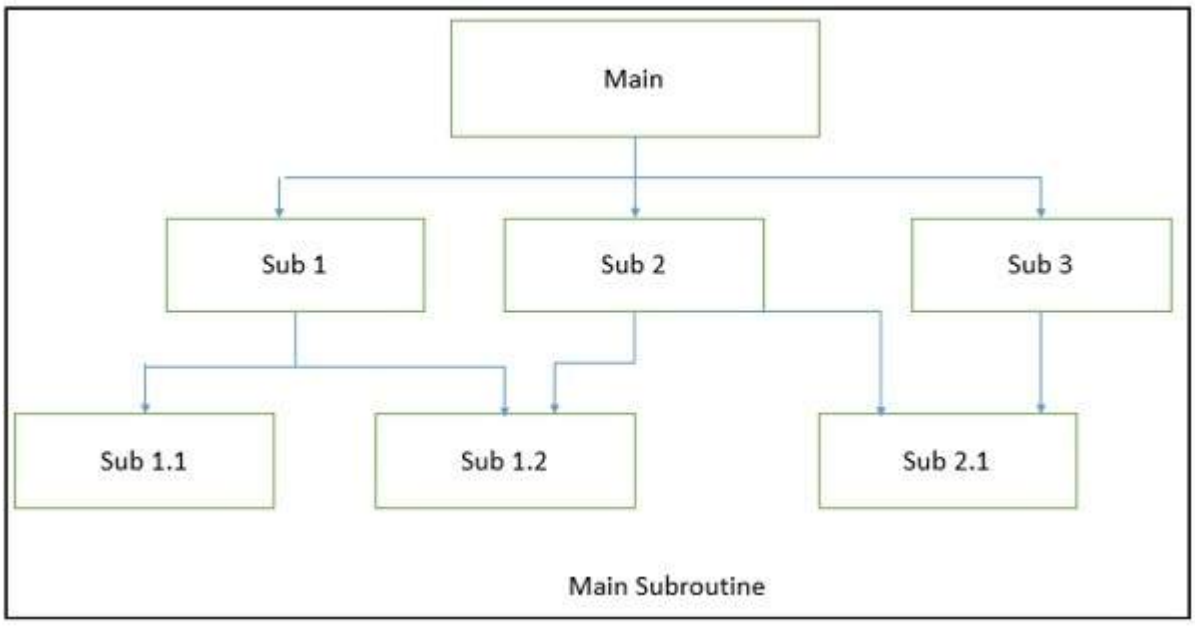
Main-subroutine

The aim of this style is to reuse the modules and freely develop individual modules or subroutine. In this style, a software system is divided into subroutines by using top-down refinement according to desired functionality of the system.

These refinements lead vertically until the decomposed modules is simple enough to have its exclusive independent responsibility. Functionality may be reused and shared by multiple callers in the upper layers.

There are two ways by which data is passed as parameters to subroutines, namely −

- **Pass by Value** − Subroutines only use the past data, but can’t modify it.
- **Pass by Reference** &minu; Subroutines use as well as change the value of the data referenced by the parameter.



Advantages

Significant advantages of Main-subroutine are — it is easy to decompose the system based on hierarchy refinement. Further, it can be also used in a subsystem of object oriented design.

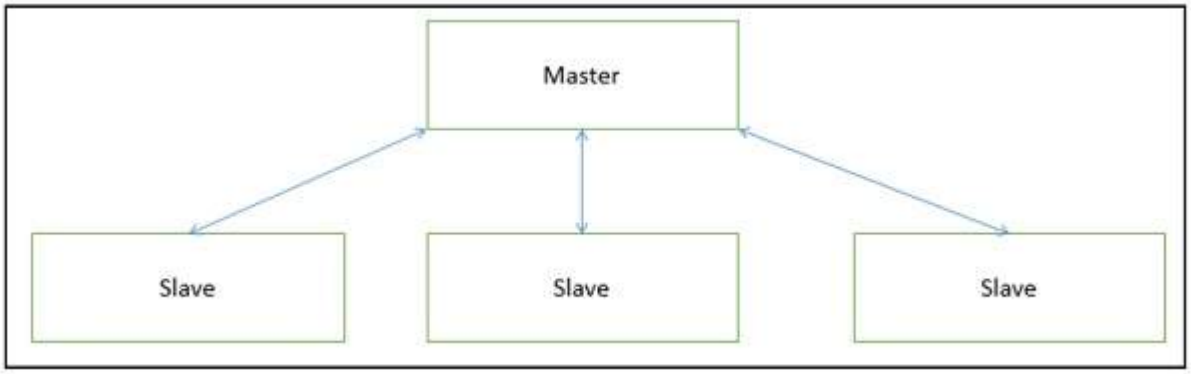
Disadvantages

Main-subroutine has some disadvantages such as tight coupling may cause more ripple effects of changes. Secondly, it contains globally shared data, so it is also vulnerable.

Master-Slave

This approach applies the 'divide and conquer' principle and supports fault computation and computational accuracy. It is a modification of the main-subroutine architecture that provides reliability of system and fault tolerance.

In this architecture, slaves provide duplicat services to the master, and the master chooses a particular result among slaves by a certain selection strategy. The slaves may perform the same functional task by different algorithms and methods or totally different functionality. It includes parallel computing in which all the slaves can be executed in parallel.



The implementation of the Master-Slave pattern follows five steps −

- **Step 1** − Specify how the computation of the task can be divided into a set of equal sub-tasks and identify the sub-services that are needed to process a sub-task.
- **Step 2** − Specify how the final result of the whole service can be computed with the help of the results obtained from processing individual sub-tasks.
- **Step 3** − Define an interface for the sub-service identified in step 1. It will be implemented by the slave and used by the master to delegate the processing of individual sub-tasks.
- **Step 4** − Implement the slave components according to the specifications developed in the previous step.
- **Step 5** − Implement the master according to the specifications developed in step 1 to 3.

Applications

- Suitable for applications where reliability of software is critical issue.
- Widely applied in the areas of parallel and distributed computing.

Advantages

- Faster computation and easy scalability.
- Provides robustness as slaves can be duplicated.
- Slave can be implemented differently to minimize semantic errors.

Disadvantages

- Communication overhead.
- Not all problems can be divided.
- Hard to implement and portability issue.

Virtual Machine Architecture

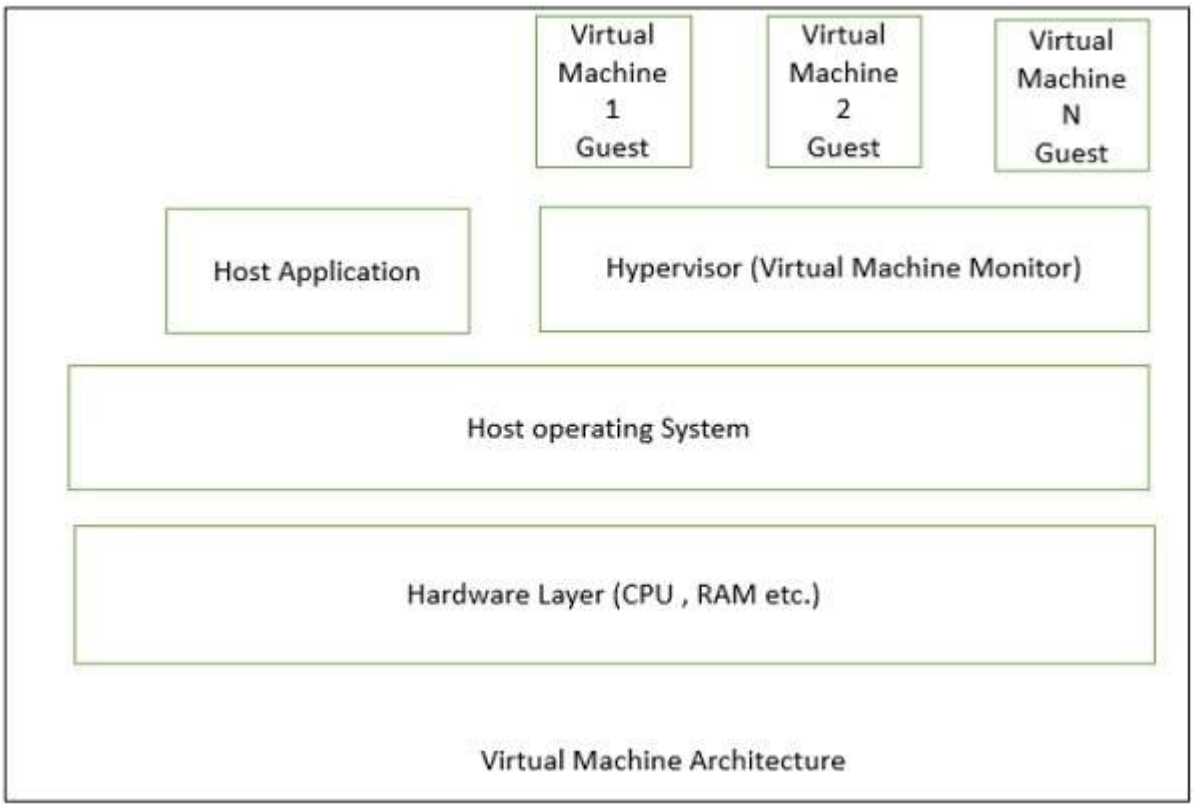
Virtual Machine architecture pretends some functionality, which is not native to the hardware and/or software on which it is implemented. A virtual machine is built upon an existing system and provides a virtual abstraction, a set of attributes, and operations.

In virtual machine architecture, the master uses the ‘same’ subservice’ from the slave and performs functions such as split work, call slaves, and combine results. It allows developers to simulate and test platforms, which have not yet been built, and simulate "disaster'' modes that would be too complex, costly, or dangerous to test with the real system.

In most cases, a virtual machine splits a programming language or application environment from an execution platform. The main objective is to provide **portability**. Interpretation of a particular module via a Virtual Machine may be perceived as −

- The interpretation engine chooses an instruction from the module being interpreted.
- Based on the instruction, the engine updates the virtual machine’s internal state and the above process is repeated.

The following figure shows the architecture of a standard VM infrastructure on a single physical machine.



The **hypervisor,** also called the **virtual machine monitor**, runs on the host OS and allocates matched resources to each guest OS. When the guest makes a system-call, the hypervisor intercepts and translates it into the corresponding system-call supported by the host OS. The hypervisor controls each virtual machine access to the CPU, memory, persistent storage, I/O devices, and the network.

Applications

Virtual machine architecture is suitable in the following domains −

- Suitable for solving a problem by simulation or translation if there is no direct solution.

- Sample applications include interpreters of microprogramming, XML processing, script command language execution, rule-based system execution, Smalltalk and Java interpreter typed programming language.
- Common examples of virtual machines are interpreters, rule-based systems, syntactic shells, and command language processors.

Advantages

Because of having the features of portability and machine platform independency, it has following advantages −

- Simplicity of software development.
- Provides flexibility through the ability to interrupt and query the program.
- Simulation for disaster working model.
- Introduce modifications at runtime.

Disadvantages

- Slow execution of the interpreter due to the interpreter nature.
- There is a performance cost because of the additional computation involved in execution.

Layered Style

In this approach, the system is decomposed into a number of higher and lower layers in a hierarchy, and each layer has its own sole responsibility in the system.

Each layer consists of a group of related classes that are encapsulated in a package, in a deployed component, or as a group of subroutines in the format of method library or header file.

Each layer provides service to the layer above it and serves as a client to the layer below i.e. request to layer i +1 invokes the services provided by the layer i via the interface of layer i. The response may go back to the layer i +1 if the task is completed; otherwise layer i continually invokes services from layer i -1 below.

Applications

Layered style is suitable in the following areas −

- Applications that involve distinct classes of services that can be organized hierarchically.
- Any application that can be decomposed into application-specific and platform-specific portions.
- Applications that have clear divisions between core services, critical services, and user interface services, etc.

Advantages

It has following advantages −

- Design based on incremental levels of abstraction.
- Provides enhancement independence as changes to the function of one layer affects at most two other layers.
- Separation of the standard interface and its implementation.
- Implemented by using component-based technology which makes the system much easier to allow for plug-and-play of new components.
- Each layer can be an abstract machine deployed independently which support portability.

- Easy to decompose the system based on the definition of the tasks in a top-down refinement manner
- Different implementations *withidenticalinterfaces* of the same layer can be used interchangeably

Disadvantages

Since, many applications or systems are not easily structured in a layered fashion, so it has following disadvantages −

- Lower runtime performance since a client's request or a response to client must go through potentially several layers.
- There are also performance concerns on overhead on the data marshaling and buffering by each layer.
- Opening of interlayer communication may cause deadlocks and "bridging" may cause tight coupling.
- Exceptions and error handling is an issue in the layered architecture, since faults in one layer must spread upwards to all calling layers