**Batch: C1 Roll No.: 16010121041**

**Experiment / assignment / tutorial No 2**

|  |
| --- |
| **TITLE:** Architecture of any operating system, draw diagram using ADL |

**AIM:** Study of architecture of operating system.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**Using ADL represent operating system architecture**

**CO 1.** Design the architecture of software systems in various architectural styles.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

* <https://www.xcubelabs.com/blog/software-architecture-understanding-styles-and-patterns-for-effective-system-design/#:~:text=Architectural> Styles are high-level strategies that provide an,or homes. Examples include Layered Event-Driven and Microservices. Last visited on 24 July 2024

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Background Theory:**

A software system's high-level structure, or software architecture, is made up of the rules, patterns, and guidelines that determine how the system is organized, interacts, and is related to its components.

The design decisions will help in providing:

Modularity: allowing the system into interchangeable components that can be developed, tested, and maintained independently.

Encapsulation: Hiding the details of the components, exposing only the necessary information, thus reducing the complexity of the system.

Security: Incorporating the measures to protect the system against the unauthorized access.

Documentation: Provides clear documentation of the system architecture, thus facilitating communication and better understanding of the system.

Performance: Ensuring that the system meets the required performance metrics such as resource utilization, throughput, etc.

There are various types of Software Architectural Styles.

* Data-centered architecture
* Data-flow architecture
* Call and return architectures
* Object-oriented architectures
* Layered architectures

Different architectural patterns will be used for implementing software systems based on:

1. System Requirements

Functional Requirements: What the system needs to do, the features and capabilities it must provide.

Non-Functional Requirements: Attributes such as performance, scalability, security, maintainability, and usability.

**2. Design Principles**

Separation of Concerns: Dividing the system into distinct sections, each addressing a separate concern.

Modularity: Designing the system in a way that components can be developed, tested, and understood in isolation.

Abstraction: Hiding the complexity of implementation details, providing a simplified interface.

Encapsulation: Bundling data and methods that operate on the data within one unit, restricting access to some of the object's components.

**3. Scalability and Performance**

The ability to handle increased load, either by scaling up (adding more resources) or scaling out (adding more nodes).

Patterns like microservices and event-driven architecture are designed to improve scalability and performance.

**4. Maintainability and Flexibility**

Ease of modifying the system to fix bugs, add features, or adapt to changing requirements.

Patterns like layered architecture and component-based architecture promote maintainability and flexibility.

**5. Security**

Protecting the system from unauthorized access and ensuring data integrity and confidentiality.

Patterns like service-oriented architecture (SOA) and microservices can help isolate and secure individual components.

**6. Development Team Skills and Experience**

The expertise of the development team with certain technologies and architectural patterns.

Patterns should align with the team's capabilities to ensure effective implementation and maintenance.

**7. Technology Stack**

The programming languages, frameworks, and tools that will be used in the system.

Patterns should be chosen to leverage the strengths of the chosen technology stack.

**8. Integration Requirements**

How the system will interact with other systems, services, and databases.

Patterns like microservices and SOA facilitate integration with external systems and services.

**9. Deployment Environment**

Whether the system will be deployed on-premises, in the cloud, or in a hybrid environment.

Patterns like cloud-native architecture are optimized for cloud deployment.

**10. Business Goals**

Alignment with the strategic objectives of the organization, including time-to-market, cost constraints, and long-term vision.

Patterns should support achieving the business goals effectively.

**11. User Experience**

The expectations of end-users in terms of interface design and interaction.

Patterns like model-view-controller (MVC) help create a better user experience by separating concerns.

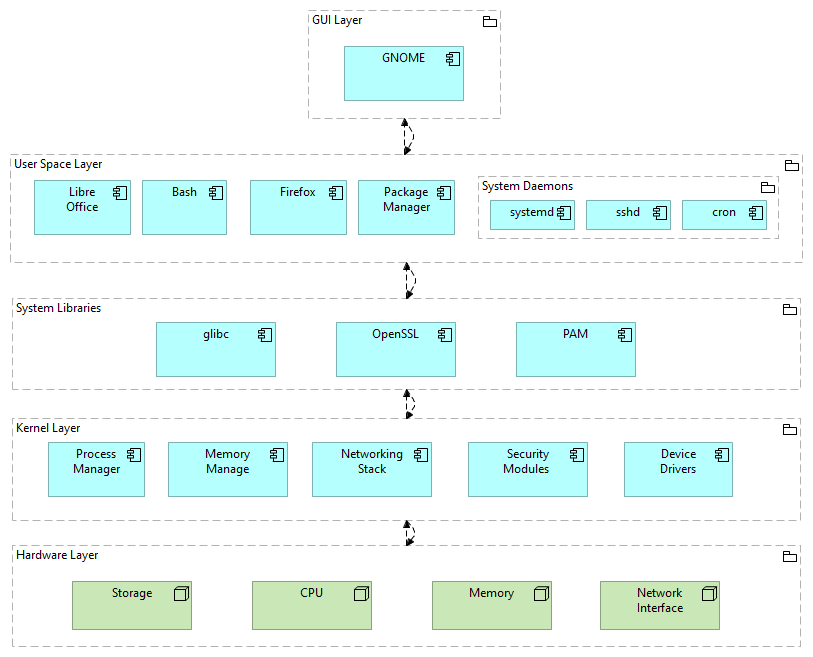
Software architecture patterns offer reusable designs for various situations, to offer advantages such as improved efficiency, productivity, speed, cost optimization, and better planning.

**Laboratory Work**:

The previous experiment introduces ArchiMate a tool used for drawing Software architecture design. The students will be required to study architecture of any of the Operating System from the following list and draw diagram

* Ubuntu
* Windows
* macOS

**LINUX architecture**



Post Laboratory questions:

1. Compare Software Architectural patterns and Design Pattern

**Software Architectural Patterns** provide a high-level blueprint for organizing a system's components and their interactions. They focus on the overall structure and relationships between modules.

**Design Patterns** are reusable solutions to common design problems within a specific context. They focus on the design of individual classes or objects and their interactions.

1. Explain Various type of Software Architecture Patterns giving example of each

**Layered Architecture**: Organizes components into layers, with each layer interacting only with the layer above or below it. (Example: web applications with presentation, business logic, and data access layers)

**Client-Server Architecture**: Separates the application into a client and a server. The client sends requests to the server, which processes them and sends back responses. (Example: web browsers and web servers)

**Event-Driven Architecture**: Reacts to events and triggers corresponding actions. (Example: user interface components responding to user input)

**Microservices Architecture**: Breaks down an application into small, independent services that communicate via APIs. (Example: modern cloud-based applications)

**Peer-to-Peer Architecture**: Allows nodes in a network to communicate directly with each other. (Example: file-sharing networks)

1. Explain in brief the architecture of Android Operating system.

**Applications**: The top layer, containing user-facing apps.

**Application Framework**: Provides APIs for developers to build apps.

**Libraries**: Contains various libraries for core functionalities like graphics, networking, and database.

**Android Runtime**: Handles the execution of Dalvik or ART virtual machines.

**Linux Kernel**: The foundation of the system, providing low-level services like memory management, process scheduling, and device drivers.