Unit 6: Abstractions for programming

1. Abstractions Levels
2. Libraries
3. System Software
4. Toolkits
5. Higher Programming Languages
6. Object -oriented approaches

#Past Questions

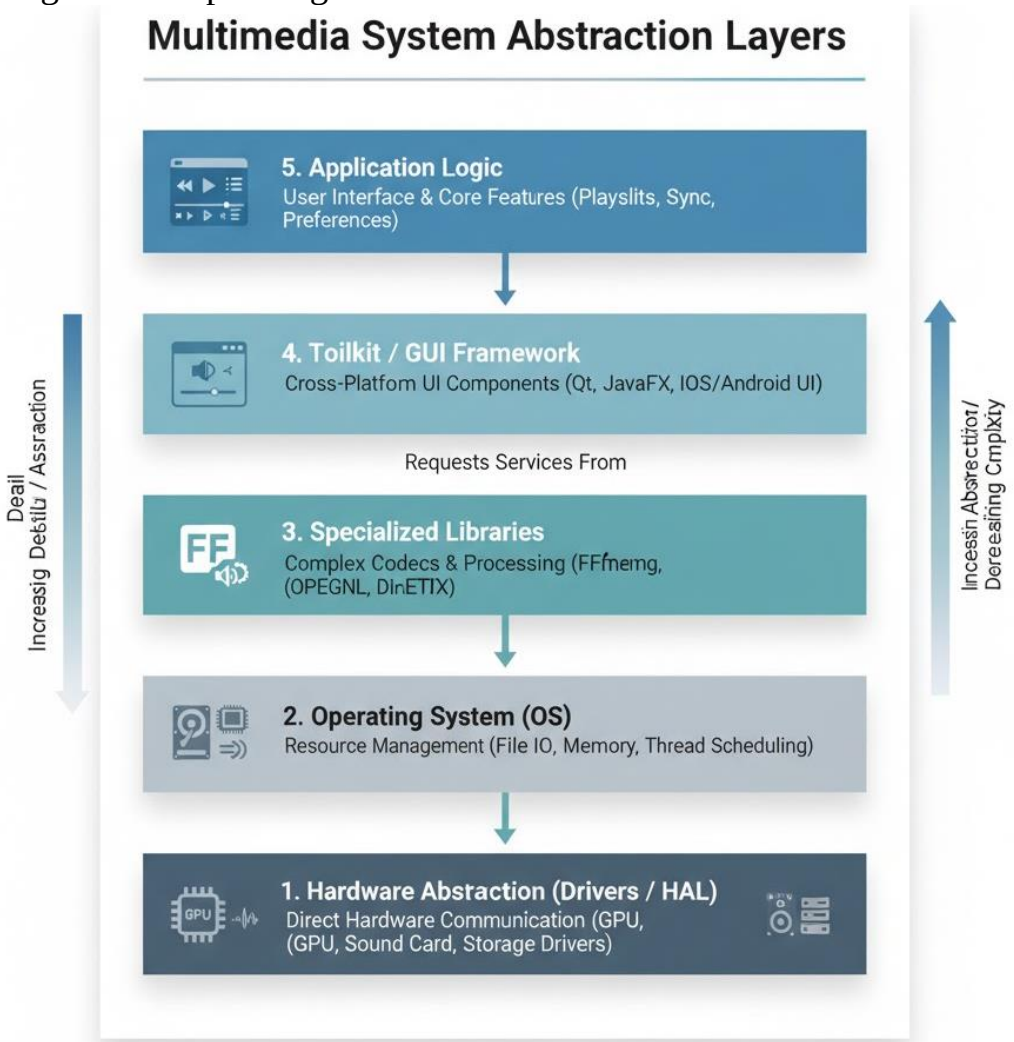
2024 Q10: What do you mean by abstraction levels in multimedia? Explain each level in detail with an example. [4+6]

1. Abstraction Levels

Abstraction is a programming concept that hides the complex implementation details and exposes only the necessary functionality to the user.
In multimedia, abstraction **simplifies programming** and helps manage **real-time media** (audio, video, text, images).
Example: Clicking "Play" on a video doesn't require knowing how decoding or syncing works.
Why Abstraction?

- Reduces complexity
- Increases code readability
- Supports modular design

Abstraction levels in programming define different approaches with a varying degree of detail for representing, accessing and manipulating data.



Levels of Abstraction in Programming:

1. Hardware Abstraction

- **Purpose:** Hides hardware details such as sound cards, graphic cards, and cameras.
- **Role:** Provides a unified interface to communicate with hardware via **drivers or APIs**.
- **Example:** DirectX, OpenGL, OpenAL, device drivers.

2. OS-Level Abstraction

- **Purpose:** Manages system-level operations needed for multimedia.
- **Role:** Handles **files, memory, I/O, and thread management** for smooth playback and recording.
- **Example:** Windows, Linux multimedia APIs.

3. Library Abstraction

- **Purpose:** Provides ready-made functions to handle multimedia data.
- **Role:** Encodes, decodes, compresses, and synchronizes audio/video streams.
- **Example:** FFmpeg, SDL, OpenCV.

4. Toolkit Abstraction

- **Purpose:** Provides **Graphical User Interface (GUI)** components for user interaction.
- **Role:** Controls buttons, sliders, progress bars, and visual timelines.
- **Example:** Qt, JavaFX, GTK.

5. Application Abstraction

- **Purpose:** Contains the **main logic** of the multimedia application.
- **Role:** Coordinates all lower layers — handling playback, synchronization, volume, and user commands.
- **Example:** VLC Player logic, media synchronization, playlist handling.

Example: Playing a Video File

Abstraction Level	What Happens in the Video Player Example
1. Hardware Abstraction	Your app doesn't directly talk to GPU, sound card, or storage — it uses drivers (e.g., sound card driver).
2. OS Abstraction	The player asks the operating system to open the file, allocate memory, and manage threads for audio/video.
3. Library Abstraction	It uses FFmpeg to decode the video and audio streams instead of writing decoding logic from scratch.
4. Toolkit Abstraction	The GUI (buttons like play, pause, slider, volume) is made using a toolkit like Qt or JavaFX.
5. Application Abstraction	The app logic controls video sync, handles playlists, remembers where the user stopped watching, etc.

2. Libraries

A library is a collection of **precompiled code** that can be used by programs to perform specific tasks without writing code from scratch.

It helps perform specific tasks — such as image processing, sound playback, or video compression — without coding everything from scratch.

Benefits:

- Saves development time
- Improves code reliability
- Encourages reusability

Examples:

- OpenCV – for image/video processing.
- FFmpeg – for audio/video compression and conversion.
- SDL – for handling graphics, sound, and input devices.

---

3. System Software

System software is responsible for managing and controlling multimedia hardware (like speakers, cameras, and graphic cards) and software resources.
It integrates multimedia device access **directly into the operating system**, so programs can easily interact with multimedia devices **without using separate libraries**.

Example :
When you play a video on Windows:

- The **OS (system software)** handles reading the video file, decoding it, synchronizing audio/video, and sending output to the display and speakers.

Data as Time Capsules

- "Time capsule" data contains both **the content** and **the time during which it is valid**.
- It is like a "package" that carries data + its timing information.
- Common in **video**, where each frame has a **timestamp**.

Example

- A video file stores each frame with a start and end time.
- When played, the video uses these timestamps to display each frame in the correct sequence.
- This ensures **synchronization** between audio and video.

Analogy

Think of a **photo slideshow**:
Each photo (data) is shown for 5 seconds (time capsule = photo + 5s).

Data as Streams

- A **stream** is a **continuous flow of data** over time — often used in **audio** and **video playback**.
- Data is not all loaded at once; it flows bit by bit (like a river).
- Streams can be **played, paused, rewound, or forwarded**.

Example

- **Audio Stream in Music Player:** The music player receives chunks of audio data continuously from a source (like a server) and plays it in real-time.
- You can **play, pause, skip, or rewind** the stream.

---

4. Toolkits

A **toolkit** is a collection of pre-written software components that help developers build multimedia applications easily. It provides **ready-to-use functions** for handling **graphics, audio, video, and animation** without dealing with low-level hardware details.

Key Points

- **Pre-built Components:** Includes built-in tools for audio/video playback, image display, and graphics rendering.
- **Cross-Platform:** Works across multiple operating systems (Windows, macOS, Linux).
- **Abstraction:** Hides complex multimedia operations and offers simple API functions to developers.

Examples

1. **SDL (Simple DirectMedia Layer):** Used for developing games and multimedia apps that need graphics, sound, and input handling.
2. **GStreamer:** Framework for building streaming and real-time video/audio processing applications.
3. **OpenCV (Open Source Computer Vision Library):** Used for image and video processing, especially in **computer vision** and **AI-based applications**.

Types of Toolkits:

- **GUI Toolkits**: Provide components like buttons, windows, and sliders
- **Web Development Toolkits**: Contain tools for routing, session handling, templates
- **Data Science Toolkits**: Help in visualization, data cleaning, etc.

Popular Toolkits:

- **Tkinter** (Python GUI)
- **Qt Toolkit** (C++ GUI)
- **ReactJS Toolkit** (JavaScript UI)
- **Pandas + Matplotlib** (Python Data Science)

---

5. Higher Programming Languages

In the context of multimedia systems, higher programming languages (HLLs) are used to process continuous media data like audio, video, and text. These languages simplify the interaction with multimedia data and hardware, often abstracting hardware details and device driver interactions.

Here's how media is handled in HLLs:

1. Media as Types

- Multimedia elements like audio, video, or images are treated as **data types** in programming.
- This allows direct manipulation of multimedia data, similar to integers or strings.

  Example:
  In OCCAM-2,

  ldu.left1, ldu.left2 → left audio channels
  ldu.left_mixed → combined/mixed audio output

You can easily perform operations like adding or mixing two audio streams.

2. Media as Files

- Multimedia data is treated like **files** that can be opened, read, written, and processed using normal file operations.
- **Example:**

  file_h1 = open(MICROPHONE_1)
  file_h2 = open(MICROPHONE_2)
  file_h3 = open(SPEAKER)
  read(file_h1)
  read(file_h2)
  mix(file_h3, file_h1, file_h2)
  *activate(*file_h1, file_h2, file_h3)

Here, data from two microphones is read, mixed, and output through the speaker file.

3. Media as Processes

- Multimedia elements are handled by **processes** that run concurrently.
- Each process manages part of the media (e.g., one for audio, another for video) and communicates in real time.
- **Example:**
  PROCESS_1 records audio while PROCESS_2 plays video — both synchronized for real-time playback.

4. Programming Language Requirements

- **Interprocess Communication (IPC) Mechanism:**
  - Enables processes to share and synchronize multimedia data.

- **Timing & Synchronization:** The programming language must support an IPC mechanism that handles the transmission of continuous media data like audio and video in a timely manner.
- **Example:** An IPC mechanism must allow one process to send an audio stream to another process that synchronizes the audio playback with a video process. Delays in data transmission could cause audio and video to get out of sync.

- **Language Extensions**

  - Instead of new languages, existing HLLs are **extended** with multimedia features.
  - These extensions add support for **real-time data**, **parallel processing**, and **media synchronization**.
  - **Example:** Extensions in C or ADA could include multimedia libraries or real-time scheduling features.

---

6. Object-Oriented Approaches

Object-Oriented Programming (OOP) is a programming paradigm based on **objects**, which combine **data (attributes)** and **methods (functions)**.
In multimedia systems, OOP helps model complex data like **audio, video, text, and graphics** using object-based abstraction — making multimedia components modular, reusable, and easy to manage.

1. Class

- A **class** is a **blueprint** or **template** for creating objects.
- It defines the structure (data members) and behavior (methods) that its objects will have.
- Acts like a user-defined data type.

Example(java):

```
class Person {
   String name;
   int id;
   void showData() {
      System.out.println("Name: " + name + ", ID: " + id);
   }
}
```

2. Object

- An **object** is an **instance of a class**.
- It has:
  - **Identity** – unique reference (e.g., object name)
  - **State** – current values of its attributes
  - **Behavior** – actions it can perform (methods)

Example:

```
Person p1 = new Person();
p1.name = "Raj";
```

```
p1.id = 101;
p1.showData();
```

3. Inheritance

- **Inheritance** allows one class (**child/derived**) to acquire properties and behaviors of another (**parent/base**) class.
- It promotes **code reusability** and supports **hierarchical relationships**.

Example:

- Base Class → Bird
- Derived Class → FlyingBird
- Subclass → Robin
  Here, a **Robin "is a" Bird**, showing an *is-a* relationship.

4. Polymorphism

- **Polymorphism** means "many forms."
- It allows the same method or operation to behave differently based on the object that calls it.
- Achieved through **function overloading** or **method overriding**.

Examples:

- **Operator Overloading:**
  - 5 + 3 → 8 (integer addition)
  - "Hello" + "World" → "HelloWorld" (string concatenation)
- **Method Overriding:**
  - findArea() behaves differently in Circle and Square classes.

Why Use OOP?

- Code reusability
- Easier maintenance
- Better data security
- Organized code structure

Class Types in Multimedia Applications

Type	Description	Example
Application-Specific Metaphors	Custom classes for specific apps	`class Track {}`
Application-Generic Metaphors	Reusable UI elements	`class Button {}`
Devices	Represent hardware	`class Microphone {}`
Processing Units	Handle data processing	`class Compressor {}`
Media	Represent media types	`class Audio {},class Video {}`
Communication Metaphors	Handle data transfer/sync	`class Stream {},class Synchronizer {}`

Distribution of BMOs and CMOs

1. BMO (Basic Multimedia Object)

- **Definition:** BMOs are the lowest-level components in a multimedia system.
- They represent raw media elements.

Examples:

- A video frame
- An audio sample
- A single image
- A text string

2. CMO (Composite Multimedia Object)

- **Definition**: CMOs are higher-level objects that are composed of multiple BMOs and sometimes other CMOs.
- They define structured or synchronized multimedia presentations.

Examples:

- A complete video scene with audio and subtitles
- A web page with embedded text, images, and animation
- A slideshow with narration

BMO vs CMO

Aspect	BMO (Basic Multimedia Object)	CMO (Composite Multimedia Object)
Definition	Raw media element	Combination of BMOs
Granularity	Fine	Coarse
Distribution	Near hardware/storage	At application level
Data Volume	Small but many	Few but large
Sync Type	Real-time (e.g., frames)	Structural (e.g., slides + audio)
Example	Video frame, Audio sample	Video with audio and subtitles

Why Distribution Matters

- Efficient resource use
- Better scalability
- Real-time performance in multimedia streaming