Android Open-Source Project (AOSP): Development

Day 1: Introduction to Android and AOSP

Theory

- **Overview of the Android Operating System**
    - History and evolution of Android
    - Android architecture and components: Layers from hardware abstraction to application framework.
- **Introduction to the Linux Kernel**
    - Kernel features and development process
    - Legal constraints with device drivers
    - Kernel user interface (/proc and /sys)
    - Kernel configuration
    - Native and cross-compilation
- **Introduction to AOSP**
    - What is AOSP? How it enables custom Android development.
    - Differences between AOSP and Android
    - AOSP ecosystem and community
- **AOSP Directory Structure**
    - Key directories and their purposes
- **Changes Introduced in the Android Kernel**
    - Functional changes introduced by Google
    - Additions to the kernel
    - Mainline kernel status of patches
- **Detailed Architecture of Android**
    - From hardware level to application level
- **Booting Stages of Android**

- In-depth understanding of the booting process(bootloader), kernel, and user-space initialization.

Lab 1.1

- **Setting Up the Development Environment**
  - Installing Android Studio and SDK
  - Cloning the AOSP repository
  - Using Android-specific tools
- **Hands-on with AOSP Build**
  - Practice building and running AOSP
  - Troubleshooting common build issues
- **Using the Android Emulator**
  - Compile and boot an Android Kernel
  - Extract patches from the Android Kernel

Lab 1.2

- **Exploring Sandboxing and App Communication**
  - Create a simple Android app that requests permissions (e.g., location or camera).
  - Analyze app isolation by inspecting /data/data directory.
  - Use Binder IPC to fetch data from a system service, such as LocationManager.
  - Inspect the interaction using adb logcat and analyze the logs for Binder transactions.

Lab 1.3

- **Hands-on with Android-Specific Tools**
  - **adb**:
    - Use adb devices to list devices.
    - Transfer a file to a connected device using adb push.
    - Inspect logs using adb logcat.
  - **logcat**:
    - Capture and filter logs for specific tags.

Day 2: Customizing AOSP

Theory

- **Customizing the Android System**
    - Modifying system applications
    - Customizing system settings (Customizing the build ID)
- **Introduction to Custom ROMs**
    - Building and modifying custom ROMs
- **Android Build System**
    - Makefile architecture and functions
    - Adding a new device to the build system
- **Security and Permissions in AOSP**
    - Understanding and modifying security policies
- **Bootloaders**
    - Bootloader examples and fastboot specifications
- **Developing and Debugging with ADB**
    - File transfers, package installation, logging, and debugging
- **AIDL and HIDL Concepts**
    - Introduction and differences
    - Implementation in Android

Lab 2.1

- **Hands-on with Android-Specific Tools (Continuation)**
    - **fastboot:**
        - Flash a custom boot image using fastboot flash boot <boot.img>.
    - **systrace:**
        - **Generate a system trace to analyze app performance.**
    - **perf:**
        - **Use the perf tool to analyze kernel-level performance.**

Lab 2.2

- **Customizing System Applications**
    - Modify system apps and system settings
    - Customize boot screens and build IDs
    - Rebuild and test on Raspberry Pi
- **Adding a Native Library**
    - Create and integrate an external library
    - Test on Android builds

Lab 2.3

- **Debugging the Android Platform User Space**
    - **Managed Components Debugging**:
        - Build a simple Android app with a deliberate error.
        - Use adb logcat and Android Studio debugger to identify and fix the error.
    - **Native Debugging**:
        - Write a native library in C/C++ that causes a segmentation fault.
        - Debug using gdbserver and analyze the core dump.
    - **Kernel Debugging**:
        - Boot a custom kernel on an emulator.
        - Use kgdb or inspect kernel logs via dmesg.

Day 3: Extending AOSP

Theory

- **Adding New Features to AOSP**
    - Implementing, testing, and debugging new features
- **Android Native Layer**
    - Understanding Android runtime components
    - Exploring hardware abstraction and media framework
- **SELinux Policies**
    - Overview and purpose
    - Modifying SELinux policies for custom devices

- **HAL and HAL Modifications**
    - Hardware Abstraction Layer overview
    - Customizing and extending HAL

Lab 3.1

- **Implementing a New Feature**
    - Add and test a simple new feature in AOSP
- **Using Raspberry Pi**
    - Boot Android and troubleshoot issues on Raspberry Pi
- **Device Development**
    - Add a new device to the AOSP build system
    - Explore daemons handling hardware components

Lab 3.2

- **Crash Debugging and Trace Analysis**
    - Simulate a crash in a native application by triggering a segmentation fault.
    - Analyze the crash using:
        1. adb logcat for immediate stack traces.
        2. Tombstone files (/data/tombstones) for detailed debugging.
    - Use symbols from .so files to resolve native traces.
    - Experiment with tools like **Crashlytics** to capture managed crashes.

Lab 3.3

- **Media Buffer Passing Between Components**
    - Implement a basic app that:
        - Captures video using the Camera API.
        - Encodes video using MediaCodec.
    - Analyze the use of shared memory (Ashmem) for buffer exchange.
    - Inspect logs to trace buffer flow through services like AudioFlinger or MediaCodec.

Day 4: Advanced AOSP Customization and Debugging

Theory

- **Advanced Build System Techniques**
    - Variables and compilation steps
    - Customizing the build environment
- **Android Filesystem Layout**
    - Software components installation and importance
- **Advanced Debugging with ADB**
    - Networking, remote commands, and system logs
- **HAL, Framework, and Application Interconnection**
    - Understanding the data flow and integration
- **Daemon Services**
    - Role of daemon services in Android
    - Managing and customizing daemon services
- **CTS and VTS**
    - Compatibility Test Suite overview
    - Vendor Test Suite and its importance

Lab 4.1

- **Advanced System Customization**
    - Modify "About" info and boot parameters
- **Advanced Feature Integration**
    - Add a feature requiring hardware abstraction
- **Final Testing and Debugging**
    - Full system validation on Raspberry Pi

Lab 4.2

- **Deep Dive into IPC and Shared Memory**
    - **Implement a basic Android Service.**

        - **Add an AIDL interface for inter-process communication.**
        - **Build a client app to interact with the service.**
    - **Experiment with shared memory:**
        - **Transfer large data buffers and analyze performance.**
    - **Inspect Binder transactions using the binder debugfs interface.**