

POSIX.1 conformance For Android Applications

By

Tayyaba Nafees

2011-NUST-MSPHD- CSE (E)-39

MS-11 (CSE)



Submitted to the Department of Computer Engineering in fulfillment of the requirements for the degree of

MASTER OF SCIENCE
In
SOFTWARE ENGINEERING

Thesis Supervisor

Dr. Shoab Ahmed Khan

College of Electrical & Mechanical Engineering
National University of Sciences & Technology

2012

DECLARATION

I hereby declare that I have developed this thesis entirely on the basis of my personal efforts under the guidance of my supervisor Dr. Shoab Ahmed Khan. All the sources used in this thesis have been cited and the contents of this thesis have not been plagiarized. No portion of the work presented in this thesis has been submitted in support of any application for any other degree of qualification to this or any other university or institute of learning.

TAYYABA NAFEES

Acknowledgement

This thesis would not have been conceived without the help of many, whom I owe a great deal.

First and foremost is Allah Almighty, the Most Gracious and Most Merciful who has given me the strength to read and write. Truly, we plan and he plans, And Allah is the Best of All Planners.

I would like to record my sincere gratitude to my supervisor **Prof. Dr. Shoab Ahmad Khan**, whose guidance, careful analysis and productive comments were valuable. I am grateful that he allowed me to work with him for this thesis.

I would like to thank my committee members **Dr. Usman Qamar, Dr. Mohammad Abbas, Dr. Farhan Riaz** for providing me the guidance.

I thank my parents for their lots of prayers, for allowing me to follow my ambitions for being patient with my endless years of study.

ABSTRACT

POSIX.1 conformance For Android Applications

Android operating system is designed for use in mobile computing by The Open Handset Alliance. It runs the powerful applications and gives its users a choice to select their applications and their carriers. At this time Android market has hundreds of thousands of Android applications and these applications are restricted only to the mobiles. This restriction is mainly because of portability and compatibility issues of Android operating system. So need of employing these countless Android applications on any POSIX Desktop operating system without disturbing the internal structure of application is very desirable. Besides that, it is also a developer's vital wish to double their revenue of the Android market from 53.3% to 100% by grabbing the POSIX Desktop user market as well. Thus we need to resolve these standardization and portability concerns by using POSIX standards (Portable Operating System Interface). The concepts of POSIX conformance for Android applications provide full-scale portability services and Android applications reusability for any POSIX desktop operating system. So Android applications will become usable for all POSIX desktop users in addition to mobile users. This research theme introduces POSIX.1 Android thin layer model that simply provides the POSIX conformance for Android applications. It is using the POSIX.1 APIs for Android applications, which maintains the compatibility between the POSIX Desktop operating systems and Android applications. We have done prudent analysis of this research work by implementation of the different applications in standard POSIX environment and, have verified its results. The results of POSIX.1 model clearly showed that it will boost up the Android applications market revenue up to 100% and is expected to add real-time capability, standardization and reusability.

Table of Contents

Acknowledgement	iii
ABSTRACT	iv
POSIX.1 conformance For Android Applications	iv
List of Abbreviations	ix
Chapter 1	1
1. Introduction	1
4.1. The Problem statement	2
4.1.1. Android applications portability	3
4.1.2. Is Android POSIX COMPLAINE?	3
4.1.3. Earliest Idea invention of POSIX conformance for Android	3
4.2. Contribution	4
4.2.1. Proposed ANDROID POSIX.1 Thin layer Model.....	4
4.2.2. Sparkling Improvements for Android applications.....	5
4.3. Motivation	6
4.3.1. Motivation for Android applications users	6
4.3.2. Motivation for Android developers	6
4.4. Principle of POSIX	7
8.1. Thesis structure	7
8.2. Publications	8
Chapter 2	8
2. Related work	9
2.1. Literature review	9
2.1.1. The Android.....	9
2.1.1.1. Android Features	9
2.1.1.2. Android (operating system).....	10
2.1.1.3. Android Architecture	11
2.1.1.4. The Android platform	12
2.1.1.5. Applications.....	14
2.1.1.6. Application Framework	14
2.1.1.7. Libraries.....	15
2.1.1.8. Android Runtime.....	16
2.1.1.9. Linux Kernel.....	16
2.1.1.10. Required tools.....	16
2.1.1.11. Market application restrictions.....	18
2.1.1.12. Download the Android SDK	19
2.1.1.13. Why Android?	19
2.1.1.13.1. Open.....	19
1.1.1.1.1. All applications are created equal	19
1.1.1.1.2. Breaking down application boundaries.....	20
1.1.1.1.3. Fast & easy application development.....	20

1.1.2. What POSIX Is	20
1.1.2.1. List of POSIX base standards.....	22
1.2. Chapter summery.....	24
Chapter 3	25
2. Proposed methodology.....	25
2.2. Problem statement	25
2.2.1. Android applications portability.....	26
2.2.2. Is Android POSIX COMPLAINCE?.....	27
2.2.3. Earliest Idea invention of POSIX conformance for Android.....	27
2.3. Proposed framework.....	27
2.3.1. ANDROID POSIX.1 Thin layer Model.....	27
2.1.2. Selection of POSIX.1 standard.....	32
2.1.3. Selection of binding language	32
2.1.4. Finding the POSIX Libraries.....	33
2.2. Chapter summary.....	34
Chapter 4	35
3. Implementation of ANDROID POSIX.1 Thin layer Model.....	35
3.1. The POSIX Development Environment.....	35
3.2. List of Android applications used as sample	35
3.3. Android Application template for POSIX.1.....	41
3.3.1. Used some core Portable functions.....	43
3.3.2. Opening and Closing Files functions.....	43
3.4. Sample examples code matching with Android Application template for POSIX.1	44
3.5. Tested Sample Examples	48
3.6. Chapter summary.....	56
Chapter 5	58
4. Analysis and Results	58
4.1. Functioning Outside the Standards.....	58
4.2. Quantified Feasibility analysis	59
4.2.1. Resulting impact factor for Android developers	62
4.3. Chapter summary.....	63
Chapter 6	64
5. Conclusion	64
5.1. Achievement	64
5.2. Limitations.....	65
5.3. Future work.....	65
References.....	68

List of Figures

<i>Figure. 2.1 Dalvik VM [7].....</i>	<i>11</i>
<i>Figure. 2.2 Android Architecture [8].....</i>	<i>14</i>
<i>Figure. 2.3 Android Emulator [7].....</i>	<i>18</i>
<i>Figure. 2.4 gcc-POSIX compilers for test environment.....</i>	<i>21</i>
<i>Figure. 2.5 Xcode used as test IDE.....</i>	<i>21</i>
<i>Figure. 3.1 Proposed POSIX.1 thin layer model.....</i>	<i>28</i>
<i>Figure. 3.2 Application framework diagram.....</i>	<i>29</i>
<i>Figure. 3.3 gcc compiler information used for application testing.....</i>	<i>31</i>
<i>Figure. 4.1 Text File creations, opening and closing code of sample Android POSIX.1 application</i>	<i>44</i>
<i>Figure. 5.1 Example of portable application</i>	<i>59</i>
<i>Figure. 5.2 Worldwide Smartphone Sales to End User by Operating source: Gartner (February 2013)[23].....</i>	<i>60</i>
<i>Figure. 5.3 Android market share [21].....</i>	<i>60</i>
<i>Figure.5.4 POSIX.1 thin layer Model resulting impact factors.....</i>	<i>63</i>
<i>Figure. 6.1 proposed Android OS POSIX compliant model.....</i>	<i>66</i>

List of Tables

<i>Table 2.1: SDK list [12]</i>	19
<i>Table 2.2: POSIX Standards list</i>	23
<i>Table 2.3: POSIX.1 functional commands [2]</i>	23
<i>Table 4.1: Development environment specifications</i>	35
<i>Table 4.2: Hello application framework</i>	37
<i>Table 4.3: Timer application framework</i>	38
<i>Table 4.4: File creator application framework</i>	39
<i>Table 4.5: Multithreading application framework</i>	41
<i>Table 4.6: Android POSIX.1 application template</i>	43
<i>Table 4.7: matching of POSIX.1 compliant Android application with POSXI.1 template</i>	48
<i>Table 4.8: Tested sample Android applications code and output comparison</i>	56
<i>Table 5.1: Market share analysis for Android developers [6]</i>	61
<i>Table 5.2: Comparative analysis of applications development time and manpower for Android developers</i>	62

List of Abbreviations

POSIX	Portable Operating System Interface
API	Application Programming Interface
UI	User interface
OS	Operating system
pthread	POSIX thread
VM	Virtual machine
AOSP	Android Open Source Project
IDE	Integrated development environment
AVD	Android virtual device
SDK	Software development kit
AOSP	Android Open Source Project

Chapter 1

1. Introduction

In these days Smartphone's are becoming the vital need and very soon it will be used as replacement of laptop and desktops. Mobile phone with built-in operating system is called smart phone. A Smartphone gives additional advanced computing competence plus connectivity than a simple phone.[1]. Presently, Android smart phones are becoming extra stylish by giving functionalities that once anticipated from laptop as well as desktop computing systems. [2] Using Google play Android smart phones are providing new and unimaginable functionalities to users. Mobile computing is real time computing. But mobile computing did not compete with Desktop OS because the Desktop users are still large in number plus it becomes the necessary need of user thus Mobile OS companies are still trying hard to make their space in the Desktop OS environment. Android is most famous and open source mobile operating system. It covers nearly 60% of mobile market but even Android OS (operating system) had the compatibility limitations. Therefore the need of standardization and portability is very essential. Android applications have multiple dependences so this limits the Android application utilization. One of the best possible solutions for catering these limitations is POSIX.

POSIX is a worldwide standard with a precise explanation plus a set of declarations. It is used for verifying compliance. A POSIX conformance application can easily shift from machine to machine with a very high assurance of low maintenance plus accurate operation. POSIX is the only way to go, when your desired software to run on the largest achievable collection of hardware as well as operating systems. [3]

POSIX conformance for android Application is the basic aim of this research in, which multiple Android applications are, used as sample input with the POSIX Application Programming Interface (API) standards. The research agenda based on the POSIX.1 thin layer model, which gives the POSIX conformance for Android applications. This POSIX.1 thin layer model hierarchy is:

1. Selection of POSIX standard for Android Applications (POSIX.1).
2. Need of POSIX.1 binding language.
3. Conversion of sample examples of Android in POSIX binding language and test it
4. Establishment of template for Android applications (POSIX thin layer model for Android applications)

4.1. The Problem statement

Android is open source OS introduced by Google. Android is still developing. Now a days it is becoming gradually more important to design software with an open system architecture utilizing industry adopted standards. So development of open system is dependent on these factors.

- **Inefficient usage of manpower:** Firstly, one developer deploys the whole system from zero. As the size of the project increases there is always more need of manpower.
- **Portability problem:** Secondly, software does not run in separate environment; it must co-exist with the vast amount of commercially available software and can be run on available OS.
- **Maintainability problems:** software application always required the multiple alterations at different level of development and post development.
- **Need of standardization:** lastly the biggest problem facing in these days is implementation of standards because portability and maintainability only fruitful when software developer follows the standards.

But Android performance is not enough, In addition, performance-analyzing environment has not been developed yet, and then its performance cannot be discussed well. Android OS addresses multiples challenges of today's software development process like interoperability, portability and compatibility issues. The major question is here, is Android application market is usable for all OS. Android applications standardization is major dilemma for Android market. Android applications for all OS are core idea of this research. But HOW is big question here. Thus Android applications need the openly

published standard interfaces for competing these hybrids issues in Android OS. We are applying the Android applications standardizations by using the POISX.POSIX is based on UNIX, a well-established technology. POSIX defines a standard way for an application to interface to the operating system. [4] POSIX, the Portable Operating System Interface .The goal of POSIX is the source-code portability of applications: it means transform an application from one operating system to another by simple conversion. This Thin layer model of POSIX.1 provides the portability for Android applications that can be run on any operating system.

4.1.1. Android applications portability

Android is conceded as the most popular mobile platform. Android user can use all the Google apps. There are more than 600,000 apps and games available on Google Play store. [5] But the sorrowful act is limitation of these 600,00 apps only for the Android OS. All of this work need conformance for any operating system according to the users and developers need. Because developers are also trying to employ Android in a range of other embedded systems, which have usually depend on the benefits of true real-time operating systems performance, boot-up time, real-time response, reliability, and no unseen maintenance costs.

4.1.2. Is Android POSIX COMPLAINT?

Android is considering a partial POSIX compliance. Limited POSIX threads (pthreads) library is implemented in Android Bionic library. It provides built-in support for pthreads, but implementation is very restricted. So Android applications conformance is very inspiring, which never has done yet.

4.1.3. Earliest Idea invention of POSIX conformance for Android

Android used the non standard Bionic library which restricted the android applications to only for android OS. So best into our knowledge this proposed model first time in the

history trying to merge the mobile OS Android applications with desktop POSIX OS. All this innovation has been done under the umbrella of POSIX.1 that means standardization and consistency.

4.2. Contribution

4.2.1. Proposed ANDROID POSIX.1 Thin layer Model

POSIX, the Portable Operating System Interface. The goal of POSIX is the source-code portability of applications: it means transform an application from one operating system to another by simple conversion. This goal is unattainable since most applications, especially the real-world ones, require more operating system support than you can find in any particular standard. The above unfeasible objective is now achievable through POSIX. POSIX is called useful." Useful," here, means "an aid to portability," and this brings us to the goal of POSIX: source-code portability of applications. The main intention of this work is that it will provide portability for the Android real world applications. Basically android is a Linux-based operating system designed primarily for touch screen mobile devices such as Smartphone's and tablet computers. But after the development of this thin layer model of POSIX.1.Android applications will become portable (POSIX compliance) and can be run on any operating system. This model provides the benefit to users as well as Android developers by increasing the number of users of android applications and reduces the developer time and cost because of portability and equivalence.

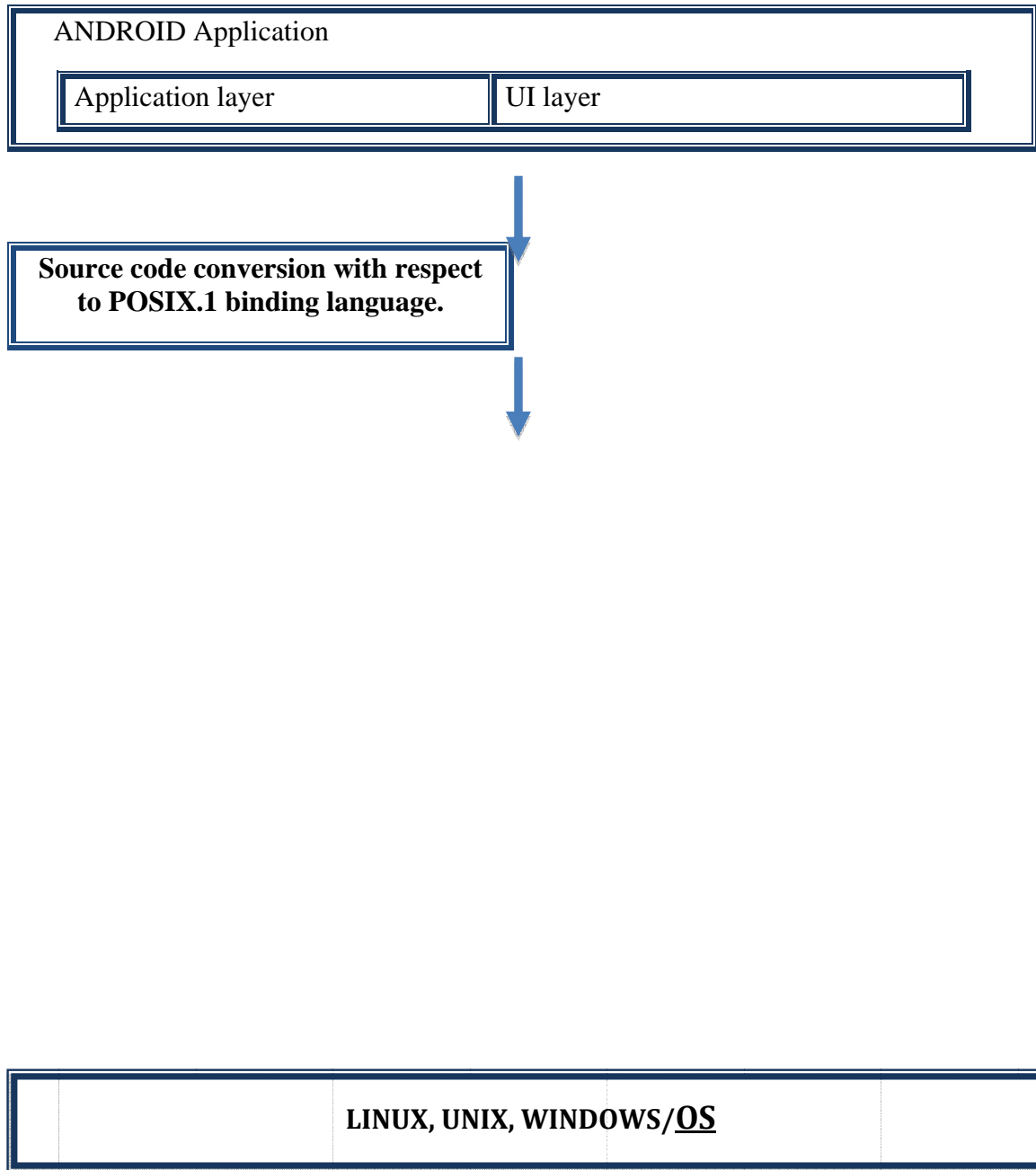


Figure 1.1 Proposed Android POSIX.1 thin layer model

4.2.2. Sparkling Improvements for Android applications

- **Android applications portability**
 - POSIX .1 thin layer model is initiation point for Android applications portability to different operating systems.

- **Android applications reusability**
 - POSIX .1 thin layer model provides the reusability of the Android applications on multiple operating systems.
- **Transformation**
 - POSIX .1 thin layer model is a standard way of transformation of application from one operation system to another with damaging the application internal structure.
- **Extend the Android market usage**
 - POSIX .1 thin layer model gives the diversity to the Android application market.

4.3. Motivation

ANDROID POSIX.1 Thin layer Model is immense development for users as well as developers because now user can use any application from Android market without concerning its mobile and desktop OS. Secondly developer needs to follow the POSIX.1 code template and its application available for mobile user plus desktop users.

4.3.1. Motivation for Android applications users

On Mobile World Congress Google ambassador said that Android growth rate is 250%, including 850,000 Android devices turn on every day. [6] We talk about usage of the Android applications user its same above. So measurement of enhancement in the Android market users including the Desktop OS users are countless and become the blast of user in the Android market.

4.3.2. Motivation for Android developers

At this point there are three basic motivational points for developer

- **Increase numbers of Android application user**
 - Android user +desktop OS user=double numbers of user

- **Double the revenue of Android developer**
 - As the numbers of user double so no doubt the revenue double. Like Android user are 3,000 apps per day and Desktop OS user 15,00 App per day so total number of apps is 4,500 now multiple developer app cost.
- **Reduce the manpower of developer**
 - Last but not least once the developer develop the app and it will be use again and again obviously it is reducing the manpower of developer

4.4. Principle of POSIX

5. POSIX is contract, interface or working like bridge between an application and the operating system. How to write applications programs or how to write operating systems is not responsibility of the POSIX.
6. The standard is written in terms of Standard C. The standard does not require that an implementation support Standard C. FORTRAN and ADA interfaces to POSIX be being developed.
7. There was no intention to specify all aspects of an operating system. Only functions used by ordinary applications are included. There are no system administration functions.
8. The standard has been absolute implementation at the source code level. But it does not give the 100% assurance that the object or binary code will run underneath a distinctive conforming implementation environment. This concerns even to two equal machines with the identical operating system.

8.1. Thesis structure

Chapter 1 This chapter provides the overview of the POSIX conformance for Android applications proposed model. Also include the problem statement in which usage of POSIX conformance for Android OS is briefly explains then contributions to our work are briefly stated.

- Chapter 2** This chapter describes the lecture review about the Android and POISX.1.
- Chapter 3** This chapter includes the proposed methodology in which selection of appropriate POSIX standard for android and programming language for POSIX is prepared.
- Chapter 4** The chapter first basically implementing POSIX.1 Thin layer Model and then looks into the Android Applications template for POSIX.1 and finally sample example code conversion and matching it with Android Applications template for POSIX.
- Chapter 5** This chapter is concerned with analysis and results of the proposed model.
- Chapter 6** This chapter presents the summery and conclusions.

8.2. Publications

- Research work of ANDROID POSIX.1 Thin layer Model are accepted in the IOSR Journal of Computer Engineering (IOSR-JCE).
- We got another acceptance notification from the 2013 9th International Conference on Natural Computation (ICNC 2013), to be held jointly with the 2013 10th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2013), from 23-25 July 2013 in Shenyang, China (impact factor 1.6).
- We also got acceptance notification from IEEE Technically Co-Sponsored Science and Information Conference 2013, London UK.

Chapter 2

2. Related work

2.1. Literature review

Basically Android is not POSIX compliant but some time it called partially POSIX compliant so this work is very restraining in lecture. Till now there is no such thing implemented for any MOBILE Operation System especially for Android. There are some software's like blue stack that provides the portability for Android applications but the concept of standardization is not applied like POSIX there and secondly all these type of software's work like application file run and exit but not gives the compatibility with underlying machine OS. May be it would be done in future for window mobile OS because of Microsoft company has its own desktop OS although in that case there is nothing for open source mobile OS. Hence there is no implementation related work. Now this chapter explains the brief history of Android OS, application development framework for Android and POSIX its standards and APIs. While the process of margining this innovative ideas is presenting in the next chapter of implementation.

2.1.1. The Android

Android is a software stack intended for mobile devices it comprises an operating system, middleware and key functions. The Android SDK offers the tools in addition to APIs essential to start developing applications lying on the Android platform via the Java programming language. [7]

2.1.1.1. Android Features

Android has the following features

- Application framework facilitating reuse plus replacement of components
- Dalvik virtual machine optimized for mobile devices
- Integrated browser stand on the open source WebKit engine

- Optimized graphics powered through a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification
- SQLite for Database
- Media sustain for common audio, video, also still image formats (MPEG4, H.264, MP3, AAC)
- Telephony GSM (hardware dependent)
- GPS, Camera, and accelerometer (hardware dependent)
- Prosperous development environment as well as a device emulator, tools in favor of debugging, memory along with performance profiling, furthermore a plug-in for the Eclipse IDE. [7]

2.1.1.2. Android (operating system)

Google introduced the Android in the mobile devices. Google Inc. Purchased the early developer of the software, Android Inc in 2005. Android's mobile operating system is established at the Linux kernel. Google as well as other partners of the Open Handset Alliance act as a team on Android's development plus release The Android Open Source Project (AOSP) is tasked among the maintenance and additional growth of Android. [8].

Android virtual machine performance is greatly improved than the java virtual machine in many perspectives similar to energy intake. [9]

The presentation of the Android circulation on 5 November 2007 was revealed among the founding of the Open Handset Alliance, a grouping of 80 hardware, software, and telecom companies dedicated to advancing open standards for mobile devices. [8]

All the Java applications running on top of a Java-based, object-oriented application framework on top of Java interior libraries running on a Dalvik virtual machine characterizing JIT compilation is part of Android. C written Libraries contain the surface manager, Open Core media framework, Bionic libc, SQLite relational database management system, WebKit layout engine, SGL graphics engine, SSL, with OpenGL ES 2.0 3D graphics API. [8]

2.1.1.3. Android Architecture

Linux kernel is used in Android for compilation. Java programming language is used for written the Android applications, and virtual machine (VM) run the applications. It's significant to note down that the VM is not a JVM as may assume, although Dalvik Virtual Machine is an open source technology. All Android application run inside an instance of the Dalvik VM, which in turn resides inside a Linux-kernel managed process, as shown below.

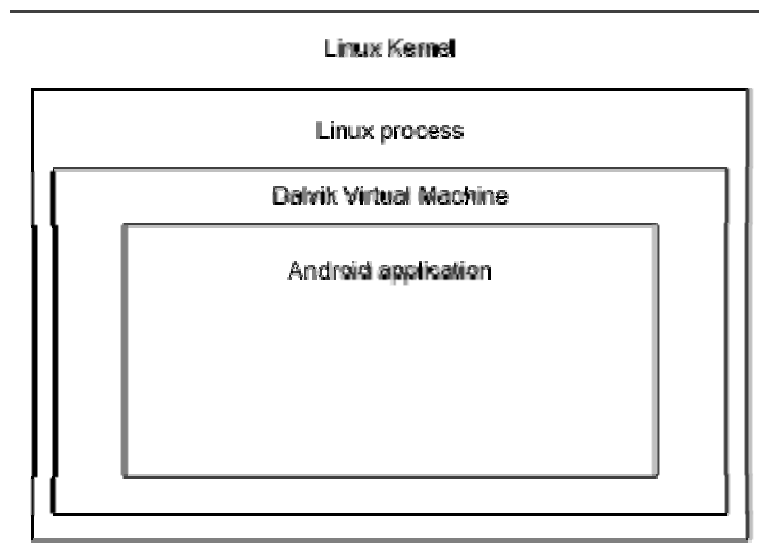


Figure. 2.1 Dalvik VM [7]

An Android application consists of one or more of the following classifications:

a. Activities

All applications with visible UI are executed with an activity. An activity is in progress, when a user selects an application from the home screen or application launcher. [7]

b. Services

For long time running application services are used, like that a network monitor and up date-checking application. [7]

c. Content providers

The content providers normally use a database server. A content provider's work is to handle approach to persisted data, like an SQLite database. If application is so straightforward, it is not essentially develop a content provider. But for building a superior application or one that offers data available to multiple activities and applications, hence accessing the data of data is responsibility of a content provider. [7]

d. Broadcast receivers

Data or respond to an event an Android application always initiated process, like that the receipt of a text message.

AndroidManifest.xml in an Android application is used for implementation. AndroidManifest.xml includes the essential configuration information to correctly install it to the device. It contains the necessary class names plus types of events the application is capable to process, in addition to the compulsory permissions the application wants to run. Such declarative safety assists decrease the probability so as to a scoundrel application can produce any harmful consequences on your device. [7]

2.1.1.4. The Android platform

Android invincible properties, it would be comfortable to mix up it with a desktop operating system. On the foundation of the Linux kernel Android is a layered environment. Android-layered environment gives the constant functionality the user interface subsystem comprises:

- Windows
- Views
- Widgets for displaying common elements such as edit boxes, lists, and drop-down lists

Android possesses a strong array of connectivity opportunities like WiFi, wireless data over a cellular connection and Bluetooth (for example, GPRS, EDGE, and 3G). The most fashionable method in Android applications is to connect to Google Maps to show an address exactly inside an application. Android software stack is also help for location-based services (such as GPS) plus accelerometers, while all Android devices are not ready with the necessary hardware. Implementation required the camera for support.

A built upon WebKit in Android provides an embeddable browser, the similar open source browser engine enriching the iPhone's Mobile Safari browser.

Traditionally, graphics/media, and data storage methods are basic fields where mobile applications are struggling. Built-in support for 2-D and 3-D graphics, including the OpenGL library in Android addresses the graphics challenges. Secondly the famous open source SQLite database is used for erased data-storage. [8]

The following diagram shows the major components of the Android operating system. Each section is described in more detail below.



Figure.2.2 Android Architecture [8]

2.1.1.5. Applications

Android equipped with a collection of basic applications like an SMS program, calendar, maps, browser, email client, contacts, and many more. Java programming language is used for writing all these applications. [10]

2.1.1.6. Application Framework

Android gives developers the capability to construct tremendously rich plus inventive applications by providing an open development platform.

It provides the countless free advantages like device hardware used, access location information, run background services, set alarms, add notifications to the status bar, along with much, much more to developer.

Developers have complete rights to the similar framework APIs used by the main applications. The concept of the reusability is very beautifully implemented here by application architecture. Like any application can broadcast its functionalities and those functionalities can be used by any other application. Component replacement also used same process. [10]

2.1.1.7. Libraries

Android comprises a collection of C/C++ libraries. These libraries are used by a variety of Android system components. Android application frameworks are used for showing these resources. Some of the core libraries are listed below:

- **System C library** – A standard C system library (libc) is implemented, which is BSD-derived implementation of the standard C system library. This library supports embedded Linux-based devices.
- **Media Libraries** - established on PacketVideo's OpenCORE; these libraries support all the popular audio and video formats as well as static image files, plus MPEG4, H.264, MP3, AAC, AMR, JPG, along with PNG.
- **Surface Manager** - manages approaches to the show subsystem as well as flawlessly multipart 2D and also 3D graphic layers from several applications
- **LibWebCore** - a new web browser engine powers both the Android browser in addition to an embeddable web view.
- **SGL** - the core 2D graphics engine
- **FreeType** - vector font rendering plusbitmap

- **SQLite** - a influential as well as lightweight relational database engine accessible to each applications [10]

2.1.1.8. Android Runtime

For developing the Android Applications developers needed the set of fundamental functionality libraries. These core libraries very similar to the java programming core libraries.

At the run time each Android application executes in one process. This process creates it particular instance in the Dalvik virtual machine. This is the beauty of Dalvik virtual machine that it can efficiently run multiple instances of the multiple applications. The Dalvik VM converts the java files into .dex format. The .dex format file is using the only footprint of memory in the device and gives the optimal result.

Linux kernel is responsible of the Dalvik VM, which gives the incredible functionality like that threading plus low-level memory management. [10]

2.1.1.9. Linux Kernel

Linux 2.6 kernel is used in the Android. Linux kernel provides the fundamental services like memory management, security, process management and network stack. Linux kernel is interface or bridge between the software stack and hardware. [10]

2.1.1.10. Required tools

For developing the Android application Android SDK and the Eclipse IDE is used. For developing the Android application developer can used any of the following operating systems.

- Microsoft Windows

- Mac OS X
- Linux

Eclipse IDE and the Android Developer Tools is standardized way to develop the Android application. Java language is used for development of the application. Instead of java VM Android used the Dalvik VM. Eclipse IDE is terrifically rich Java environment that includes code suggestion hints, context-sensitive help and many more. After the compilation of the Android code developer Toolkit attached the all required packages

The SDK is developer Toolkit that is unpacked into the mobile device of the user. The SDK have the following features:

1. Android. Jar

All the important Android SDK classes to execute the any android application are provided by the Android.jar.

2. Docs directory and Documentation.html

All the information and helpful links for development of application are given into these java documents. These java documents are available locally in the IDE and externally from the web can access easily.

3. Samples directory

Multiple sample examples with source code are available in the sample directory. These examples used multiple APIs and very helpful for the premature developer. Sample directory is idea innovation spot for starting Android application development.

4. Tools directory

adb utility (Android Debug Bridge) is example of tool directory. Android also provides the access of the command-line through command line tools.

5. Usb_driver

For testing the developed Android application the developer needs to connect the IDE with the mobile device. All the essential drivers like the G1 or the Android Dev 1 unlocked development phone for employ the connection between the device and development environment is provided by usb_driver These files are only required for developers using the Windows platform.

Android SDK port the Android applications on mobile or Emulator that used for testing the applications .the below figure shows the Emulator. [8]



Figure.2.3Android Emulator [7]

2.1.1.11. Market application restrictions

Android OS is open source but applications in the Android market are not open source. All the Android devices first of all accept the Google licensing agreement and then it

allows for downloading the application from market. Still hardware dependencies exist there. At this time more than 1.5 billions of Android application users are there. [11]

2.1.1.12. Download the Android SDK

Platform		Size	
Windows	Android-sdk_r12-windows.zip	36486190 bytes	8d6c104a34cd2577c5506c55d981aebf
	Installer_r12-windows.exe (Recommended)	36531492 bytes	367f0ed4ecd70aefc290d1f7dcb578ab
Mac OS X (Intel)	Android-sdk_r12-mac_x86.zip	30231118 bytes	341544e4572b4b1afab123ab817086e7
Linux (i386)	Android-sdk_r12-linux_x86.tgz	30034243 bytes	f8485275a8dee3d1929936ed538ee99a

Table 2.1: SDK list [12]

2.1.1.13. Why Android?

2.1.1.13.1. Open

Android provides the enormous opportunities for developers to build the marvelous applications with unbeatable qualities. The openness offers the developers to utilize the all the benefits of mobile devices. Linux kernel is base of the Android. Besides that, Dalvik VM supports them for fighting the challenges related to the memory utilization and mobile hardware resource management. Injecting new technologies benefits into the Android is only possible because of the open source OS. [13]

1.1.1.1.1. All applications are created equal

There is no difference between the 3rd party applications and Android core applications in Android devices. It means Android gives the equality for all the applications regardless of

its type. Any core application can be replaced with 3rd party application depend on the user need and demand. All the applications have equal rights in the Google market. [13]

1.1.1.1.2. Breaking down application boundaries

There are no boundaries for the developer in the Android IDE. Developer can develop any application using any core or 3rd party libraries. Sample examples are Cloud computing applications, SQLite using application. [13]

1.1.1.1.3. Fast & easy application development

Development of the application in the Android is quite easy and fast. It offers the countless built-in tools and libraries for the developer. So developer can easily make the Android applications with magnificent functionalities. [13]

1.1.2. What POSIX Is

Portable operating system interface is bridge between the two different environment applications. After using the POSIX you are basically connecting the very distinct environment applications and make the impossible task by help of it. POSIX basically dependent on:

- a. **A Compilation System:** A compiler, basically. Real live POSIX systems are supposed to support a standard language. For this purpose the compiling language is C. For getting the POSIX support in any application each system have a variety of way of compiling code, for each occurrence. For instance, under LynxOS one invokes the compiler (GNU C) with `gcc -mposix1b`, and under QNX the POSIX.4 facilities are available by default. Using the compilation system in the approved fashion makes the POSIX environment available to the program. [14]

```

Target: i686-apple-darwin11
Configured with: /private/var/tmp/llvmgcc42/llvmgcc42-2336.11~67/
2 --mandir=/share/man --enable-languages=c,objc,c++,obj-c++ --pro
11 --enable-llvm=/private/var/tmp/llvmgcc42/llvmgcc42-2336.11~67/
ple-darwin11 --with-gxx-include-dir=/usr/include/c++/4.2.1
Thread model: posix
gcc version 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 23
TAYYABAs-MacBook-Pro:Desktop tayyabanafees$ █

```

Figure.2.4 gcc-POSIX compilers for test environment

- b. **Headers:** A set of headers that defines the POSIX interface supported on the particular system. These are usually files in /usr/include, but they could be elsewhere, especially when the user are cross developing (building programs on one machine with the intention of running them on another, totally different machine); they might not even be files in the traditional sense. [14] #include <stdio.h> was used header file in given example

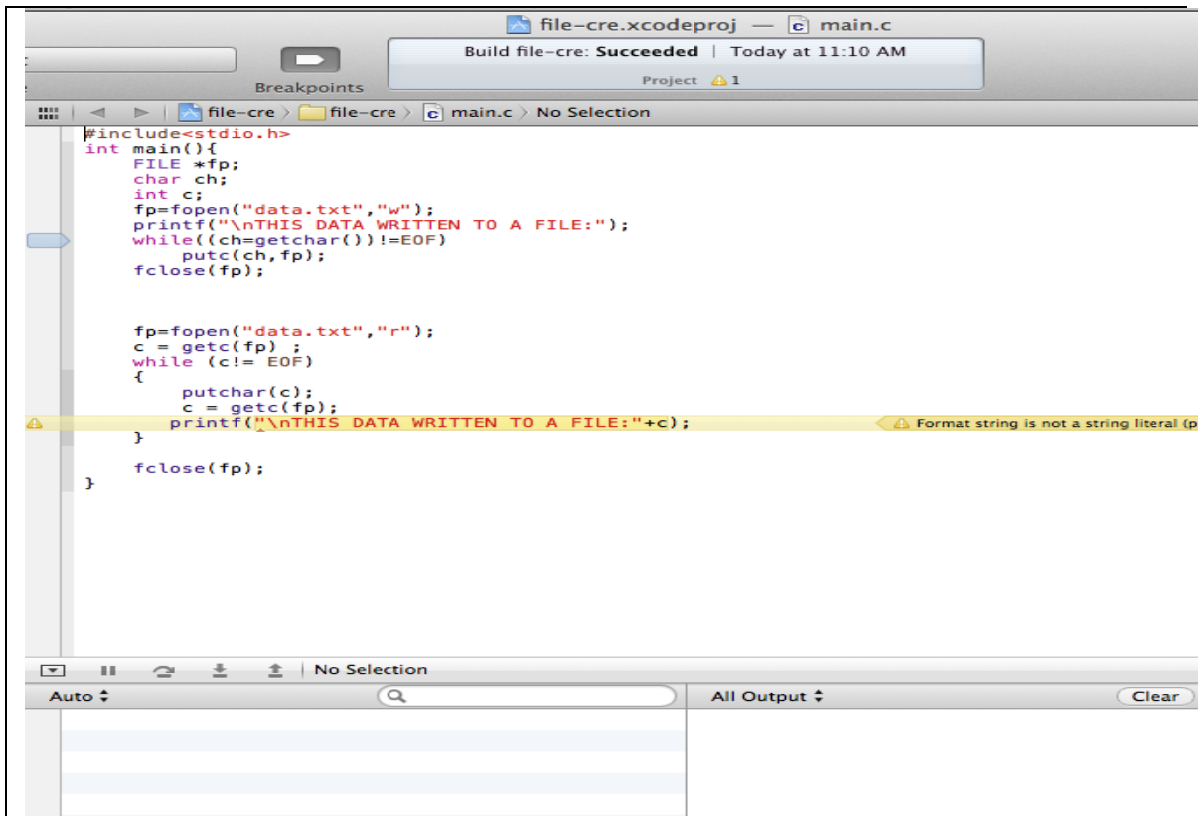


Figure.2.5 Xcode used as test IDE

- c. **Libraries:** Libraries are pre-compiled, vendor-supplied objects that implement the POSIX functionality for any one. The libraries are linked into the application when it is built, or in the case of dynamically shared libraries, when user runs the program. [14]
- d. **A Run-Time System:** Once user has built the program, the run-time, or operating system, allows him/her to run the application. For most of you UNIX folks, the runtime system is the same system under which built the application. You compile the application and then you run it, just like that. However, it's important to realize that you may compile your application in one place and run it in an entirely different environment. Especially in the real-time world, it's common to build an application in a user-friendly environment, such as SunOS on a SPARC or HP-UX on a Precision machine—or even MS-DOS (everything's relative). [14]

1.1.2.1. List of POSIX base standards.

POSIX.1	System Interface (basic reference standard) a, b
POSIX.2	Shell and Utilities
POSIX.3	Methods for Testing Conformance to POSIX, a
POSIX.4	Real-time Extensions
POSIX.4a	Threads Extensions
POSIX.4b	Additional Real-time Extensions
POSIX.6	Security Extensions
POSIX.7	System Administration
POSIX.8	Transparent File Access
POSIX.12	Protocol Independent Network Interfaces

POSIX.15	Batch Queuing Extensions
POSIX.17	Directory Services

Table 2.2: POSIX Standards list

POSIX.1 on the other hand, is not considered to be basic functionality that all systems need in order to be useful (regardless of my personal opinion). Therefore, POSIX.1 is structured as a set of well-defined options that a vendor can support, or not. The only parts of POSIX.1 that aren't optional are some additions to the basic POSIX.1 signal mechanism. POSIX.1 options.

Option Name	Functionality
<u>POSIX_PRIORITY_SCHEDULING</u>	Process scheduling control: sched_setparam, sched_getparam, sched_setscheduler, sched_getscheduler, sched_yield, sched_get_priority_max, sched_get_priority_min, sched_rr_get_interval
<u>POSIX_TIMERS</u>	Clocks and timers: clock_settime, clock_gettime, clockgetres, timercreate, timer_delete, timer_settime, timer_gettime, timer_getoverrun, nanosleep
<u>POSIX_MAPPED_FILES</u>	Files mapped as memory: mmap, munmap, ftruncate, msync (if and only if <code>_POSIX_SYNCHRONIZED_IO</code>)

Table2.3: POSIX.1 functional commands [2]

- a. **Compile-Time Checking:** the checking of an operating system's POSIX support at compile time, either from code in your application or in a totally separate conformance-checking application that you run at the same time. A generic

conformance checker is a useful program because it's not tied to a particular application. Each optional piece of POSIX comes with a constant, which defines its existence, and perhaps other symbols, which define its size and shape. Existence is defined in `<unistd.h>`, and the numeric parameters (size and shape) for each option are given in `<limits.h>`. [14]

1.2. Chapter summery

In this chapter we discussed the Android OS, its structure and Android applications basic subparts. After that we gave the information about what is POSIX its basic components and list POSIX standards. For implementing the POSIX conformance for Android application understanding of POISX.1 and Android is compulsory.

Chapter 3

2. Proposed methodology

POSIX is an international standard with an exact definition and a set of assertions, which can be used to verify compliance. A conforming POSIX application can move from system to system with a very high confidence of low maintenance and correct operation. If you want software to run on the largest possible set of hardware and operating systems, POSIX is the way to go. POSIX is based on UNIX System V and Berkeley UNIX, but it is not itself an operating system. POSIX describes the contract between the application and the operating system. POSIX does not say how to write applications programs or how to write the operating system. Instead, POSIX defines the interface between applications and their libraries. POSIX does not talk about "system calls" or make any distinction between the kernel and the user.

2.2. Problem statement

Android is open source OS introduced by Google. Android is still developing. Now a days it is becoming gradually more important to design software with an open system architecture utilizing industry adopted standards. So development of open system is dependent on these factors.

- **Inefficient usage of manpower:** Firstly, one developer deploys the whole system from zero. As the size of the project increases there is always more need of manpower.
- **Portability problem:** Secondly, software does not run in separate environment; it must co-exist with the vast amount of commercially available software and can be run on available OS.
- **Maintainability problems:** software application always required the multiple alterations at different level of development and post development.

- **Need of standardization:** lastly the biggest problem facing in these days is implementation of standards because portability and maintainability only fruitful when software developer follows the standards.

But Android performance is not enough, In addition, performance-analyzing environment has not been developed yet, and then its performance cannot be discussed well. Android OS addresses multiples challenges of today's software development process like interoperability, portability and compatibility issues. The major question is here, is Android application market is usable for all OS. Android applications standardization is major dilemma for Android market. Android applications for all OS are core idea of this research. But HOW is big question here. Thus Android applications need the openly published standard interfaces for competing these hybrids issues in Android OS. We are applying the Android applications standardizations by using the POISX.POSIX is established on UNIX, a well-established technology. POSIX defines a standard way for an application to interface to the operating system. [4] POSIX, the Portable Operating System Interface .The goal of POSIX is the source-code portability of applications: it means transform an application from one operating system to another by simple conversion. This Thin layer model of POSIX.1 provides the portability for Android applications that can be run on any operating system.

2.2.1. Android applications portability

Android is conceded as the most popular mobile platform. Android user can use all the Google apps. There are more than 600,000 apps and games available on Google Play store. [5] But the sorrowful act is limitation of these 600,00 apps only for the Android OS. All of this work need conformance for any operating system according to the users and developers need. Because developers are also trying to employ Android in a range of other embedded systems, which have usually depend on the benefits of true real-time operating systems performance, boot-up time, real-time response, reliability, and no unseen maintenance costs.

2.2.2. Is Android POSIX COMPLIANCE?

Android is considering a partial POSIX compliance. Limited POSIX threads (pthreads) library is implemented in Android Bionic library. It provides built-in support for pthreads, but implementation is very restricted. So Android applications conformance is very inspiring, which never has done yet.

2.2.3. Earliest Idea invention of POSIX conformance for Android

Android used the non standard Bionic library which restricted the android applications to only for android OS. So best into our knowledge this proposed model first time in the history trying to merge the mobile OS Android applications with desktop POSIX OS. All this innovation has been done under the umbrella of POSIX.1 that means standardization and consistency.

2.3. Proposed framework

2.3.1. ANDROID POSIX.1 Thin layer Model

POSIX, the Portable Operating System Interface .The goal of POSIX is the source-code portability of applications: it means transform an application from one operating system to another by simple conversion. This goal is unattainable since most applications, especially the real-world ones, require more operating system support than you can find in any particular standard. The above unfeasible objective is now achievable through POSIX. POSIX is called useful.” Useful," here, means "an aid to portability," and this brings us to the goal of POSIX: source-code portability of applications. The main intention of this work is that it will provide portability for the Android real world applications. Basically android is a Linux-based operating system designed primarily for touch screen mobile devices such as Smartphone’s and tablet computers. But after the development of this thin layer model of POSIX.1.Android applications will become portable (POSIX compliance) and can be run on any operating system. This model

provides the benefit to users as well as Android developers by increasing the number of users of android applications and reduces the developer time and cost because of portability and equivalence.

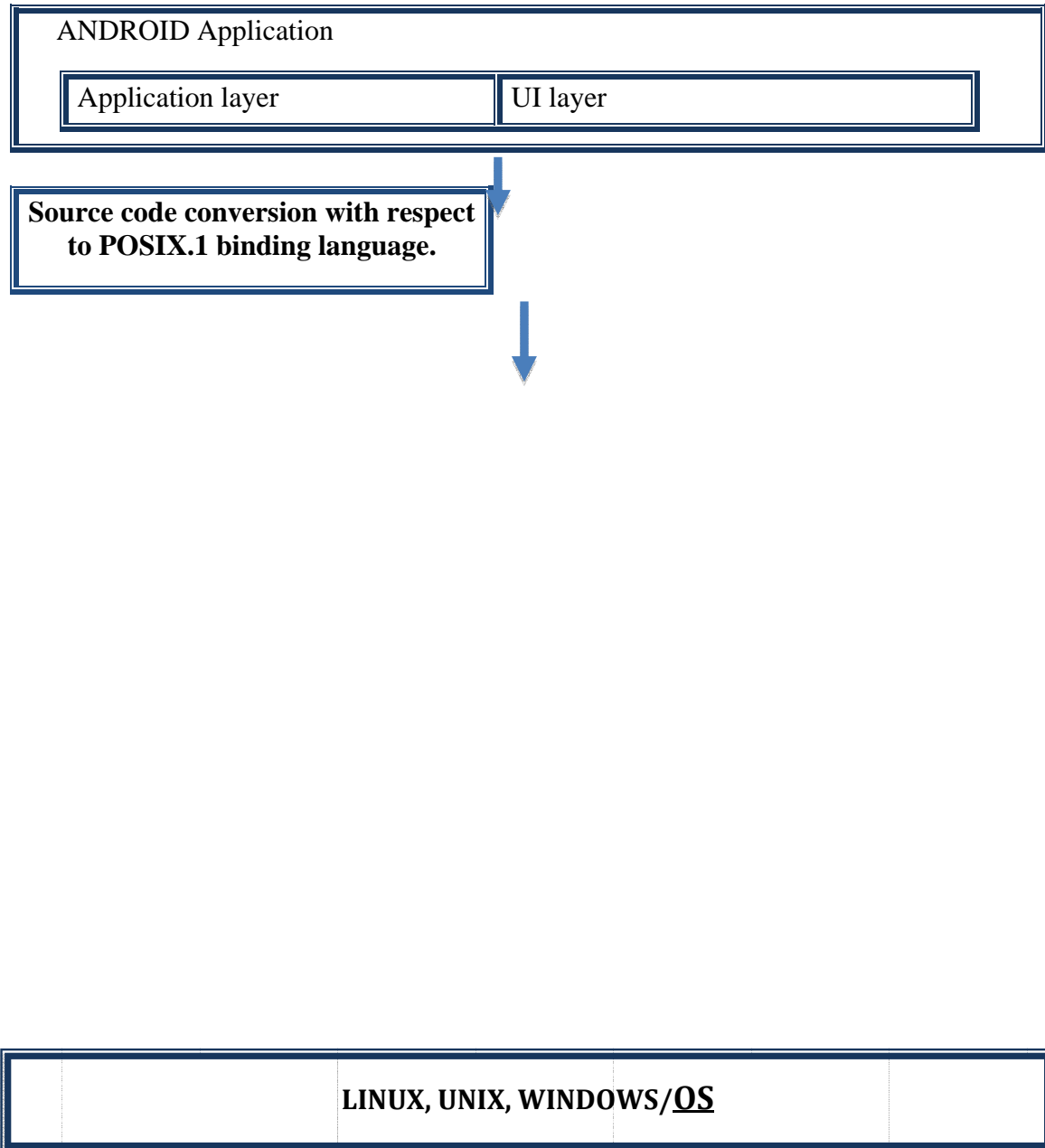


Figure.2.1 Proposed POSIX.1 thin layer model

2.1.1.1. Android application

Normally android application has two parts

- Application layer
- UI layer

Because of limitation of POSIX.1 we deal only with application framework. Application framework enables the reusability and replacement of Android components [15]. There are two main part of this framework

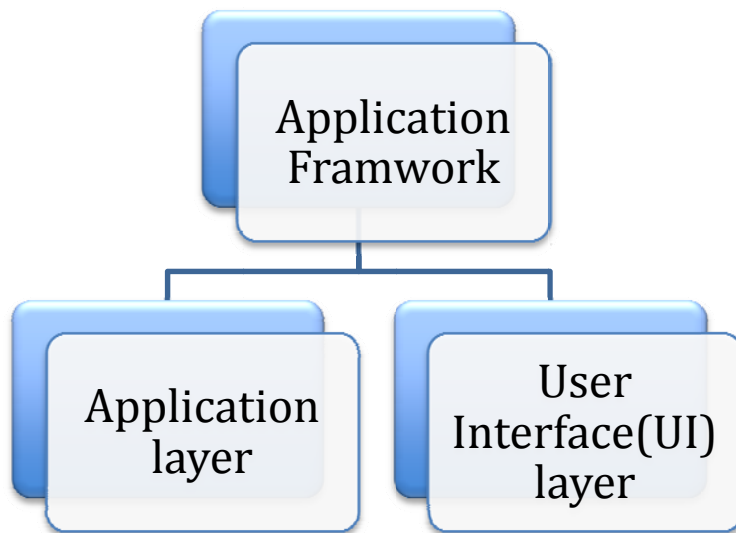


Figure.2.2 Application framework diagram

The application layer comprising:

- All the essential components of the framework is define here
- Information about the necessary features of the device
- All the application source code related to functionality of the application and behavior of the application define here.

Fundamentally application framework is key to open any Android application because it has access of all sub components of an application and offers the open development platform. [16] ANDROID POSIX.1 Thin layer Model only took the application framework of android application and send it to under layer of code conversion.

2.1.1.2. *POSIX binding language code conversion*

Basically there is no standard tool available for conversion of code according to the binding language. Secondly most of the binding language of POSIX is very extinct. Porting an existing application to run on a new system requires two major steps. These tasks can range from very easy to almost impossible. First, you have to transport the program to the target computer. Second, you have to modify the program to run in the new environment. The POSIX standard can help you in both steps. [17] There are multiple binding languages of POSIX like IEEE POSIX.5 committee is defining the ADA interface to POSIX and the IEEE POSIX.9 committee is defining the FORTRAN interface to POSIX. POSIX.1 standard is written in terms of the C programming language. POSIX supports two programming environments. One is based on the traditional C language. The other is based on the Standard C language defined by American National Standard for Information Systems—Programming Language—C, X3.159-1989. Standard C defines the C language in a more precise way and allows for more portability than traditional C. The POSIX. 1 Standard Document is dedicated to POSIX. 1, which produced an IEEE standard in 1988 and an international standard in 1990. The full legal name is: IEEE Std. 1003.1-1990 Standard for Information Technology—Portable Operating System Interface (POSIX)—PART 1. System Application Programming Interface (API) [C Language]. Today, there are many Standard C compilers on the market and most platforms support one or more of them. This research is implemented on the gcc compiler. [14] Information about compiler is:

```

Configured with: /private/var/tmp/llvmgcc42/
llvmgcc42-2336.11~67/src/configure --disable-checking --
enable-werror --prefix=/Applications/Xcode.app/Contents/
Developer/usr/llvm-gcc-4.2 --mandir=/share/man --enable-
languages=c,objc,c++,obj-c++ --program-prefix=llvm- --
program-transform-name=/^[cg][^.-]*$/s/$/-4.2/ --with-
slibdir=/usr/lib --build=i686-apple-darwin11 --enable-llvm=/
private/var/tmp/llvmgcc42/llvmgcc42-2336.11~67/dst-
llvmCore/Developer/usr/local --program-prefix=i686-apple-
darwin11- --host=x86_64-apple-darwin11 --target=i686-
apple-darwin11 --with-gxx-include-dir=/usr/include/c++/
4.2.1Thread model: posixgcc version 4.2.1 (Based on Apple
Inc. build 5658) (LLVM build 2336.11.00)

```

Figure.2.3gcc compiler information used for application testing

POSIX is a superset of standard C library.POSIX.1-2008 defers to the ISO C standard.
[18]

2.1.1.3. Compile-time checking

The POSIX compliant OS check the application POSIX compatibility or user need to tell the OS its POSIX application so firstly at compile-time define the symbols which classify it's a POSIX application. This symbol is called #define_POSIX_SOURCE.This definition tells your system that you want the POSIX definitions that were in effect for the September, 1993version of POSIX. You don't care about POSIX work that happened after that, and you don't care about anythingother than POSIX.

The _POSIX_C_SOURCE definition simply tell the compiler that you're not interested in any symbols other than those defined by POSIX or ANSI C. but don't worry about other needed symbols because its not for other all modules of the program that's work fine. This is the basic reason of modularization.

2.1.1.4. *Run-time checking*

The run-time checking is totally different from compile-time checking. Basically here we used the POSIX APIs and header files like system header and local headers file.

2.1.1.5. *Run the Android application on the POSIX compliant OS*

After the language conversion now Android application is ready for running any POSIX compliant desktop OS.

2.1.2. Selection of POSIX.1 standard

In the large list of POSIX standards selection of any one POSIX standards is very difficult but we select the POSIX.1 because it gives two type of conformance conformance-conformance to POSIX.1, C Language Binding (C Standard Language-Dependent System Support), or to POSIX. 1,C Language Binding (Common-Usage C Language-Dependent System Support). An ISO/IEC Conforming POSIX.1 Application is an application that uses only the facilities described in this standard and approved Conforming Language bindings for any ISO or IEC standard. [19]

2.1.3. Selection of binding language

Standard C language is used as the binding language for POSIX.1 but the questions are here

- Standard C language
- Difference between POSIX and Standard C Library

2.1.3.1. Why Standard C language?

The Standard C libraries are important. POSIX supplies only one part of the programming toolkit. We need the libraries provided as part of Standard C in order to write interesting programs. It is not worth wasting brain cells remembering which tools are in the Standard C box and which are in the POSIX box. It is better to remember our tools by function. This is like sorting our tools into screwdrivers and wrenches instead of Craftsman tools and Stanley tools.

2.1.3.2. Are there any Difference Between POSIX and Standard C Library?

POSIX contain a lot of things. For example, threads, semaphores, file system access API, etc. there are very minimalist of operating system libraries in the Standard C library (i.e., ANSI/ ISO C) For example, standard C library have no such function, which handle the multi threading .So, the implementation of the IPC inter process communication is restricted. It means it has no ability to tackle with multi-processing operating system. But we need the complete toolkit for implementation. POSIX is a superset of standard C library. POSIX defines a library of functions for conforming programs to use. Many of these functions are also defined in the Standard C library. Each function in the library requires you to include at least one header. This is done with a statement like:

```
#include <stdio.h>
```

Many systems support multiple development environments. How do you get the POSIX headers? You must define the symbol `_POSIX_SOURCE` before including any of the standard headers. The best way to do this is to place the statement.

```
#Define _POSIX_SOURCE 1 at the start of each file.
```

2.1.4. Finding the POSIX Libraries

The POSIX libraries are part of the standard system libraries. You can indicate that you want all vendor extensions hidden from you by defining the symbol `_POSIX_SOURCE` with the statement:

```
#Define _POSIX_SOURCE 1
```

According to the rules of Standard C, only those symbols that are in the base standard or are enabled by a specific `#define` feature test are allowed to be visible. However, many vendors require a special command to get the Standard C behavior. They include their added value by default. By defining `_POSIX_SOURCE` you should protect yourself from this added value. Every conforming POSIX system must provide a "conformance document" that describes its implementation.

2.2. Chapter summary

In this chapter we first defined the problem statement then proposed an Android POSIX.1 thin layer model to overcome the problem of portability and standardization of android applications. Next we illustrated design and working of the Android POSIX.1 thin layer model and description of this model sub components. At The end reason of selection of the appropriate POISX.1 standard and selection of standard C language for the POISX.1 also discussed here.

Chapter 4

3. Implementation of ANDROID POSIX.1 Thin layer Model

3.1. The POSIX Development Environment

POSIX provides portability at the source level. This means that you transport your source program to the target machine, compile it with the Standard C compiler using conforming headers, and link it with the standard libraries. The system vendor provides the compiler, the libraries, and headers. Strictly speaking, these are all black boxes and you do not need to know how they work. For POSIX .1 thin layer model implementation we used the following specifications.

Specifications	POSIX environment	Android Environment
OS	Macintosh	Macintosh
IDE	Xcode	Eclipse INDIDGO
Language	C	Java
Complier	gcc version 4.2.1	Java complier

Table 3.1: Development environment specifications

3.2. List of Android applications used as sample

For this model we start the implementation from very simple to the complex one like multithreading [20].

- Hello world
- Timer
- Text file creator, save data on it and display the text on the terminal
- Multithreading example [20]

The reasons of start test from very simple Android application to complex one Android application are

- Is Android application will be POSIX compliant is a question itself. So we implement the very first sample example in both environment then we move forward that why it is part of our research work.
- The User interface means graphical user interface of POISX is not very supportive for android applications
- There is no such engine or converter that convert the whole application layer of android application.
- All the gcc compiler is not POSIX and all the desktop OS are not POISX compliant
- All the implement applications are very simple in Android environment but POSIX APIs are limited in numbers. Even for hello example POSIX standard C language have specified code.
- File creator and multithreading is very important example because it used very frequent OS calls. The IEEE Std 1003.1b-1993(pp.103) also used these examples for implementation.

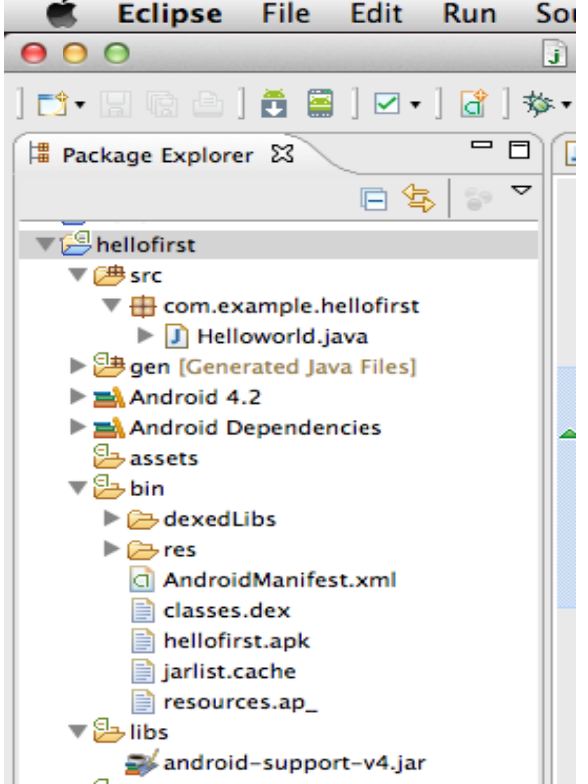
Hello world Android application layer	POISX conformance Hello world example
	<pre data-bbox="854 287 1417 630"> #define _POSIX_SOURCE 1 #include <stdio.h> int main(int argc, constchar * argv[]) { printf("%s", "Hello, World!\n"); return0; } </pre> <ul data-bbox="902 640 1417 892" style="list-style-type: none"> • % s Write the argument (assumed to be a pointer to a null-terminated character string) to the output stream. • Now it is fully portable for all language like French or Japanese also

Table 3.2: Hello application framework

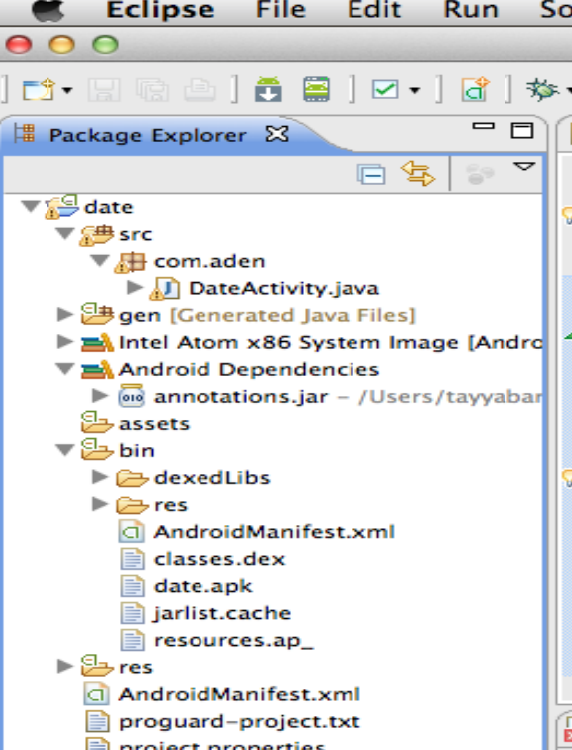
Timer Android application layer	POISX conformance Timer Android example
	<pre> #define _POSIX_SOURCE 1 #include <stdio.h> #include <time.h> main(argc, argv) int argc; char **argv; { struct tm *tmptr; time_t timer; timer = time(NULL); tmptr = localtime(&timer); printf("The current time is: \n%s", ctime(&timer)); if (tmptr ->tm_isdst) printf("Daylight savings time\n"); else printf("Standard time\n"); exit(0); } </pre>

Table 3.3: Timer application framework

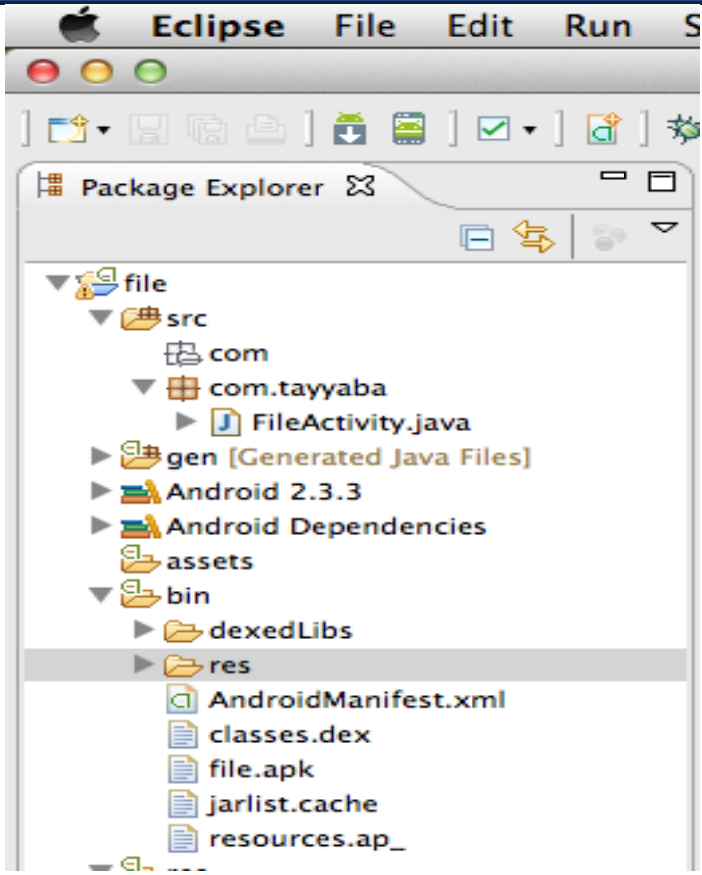
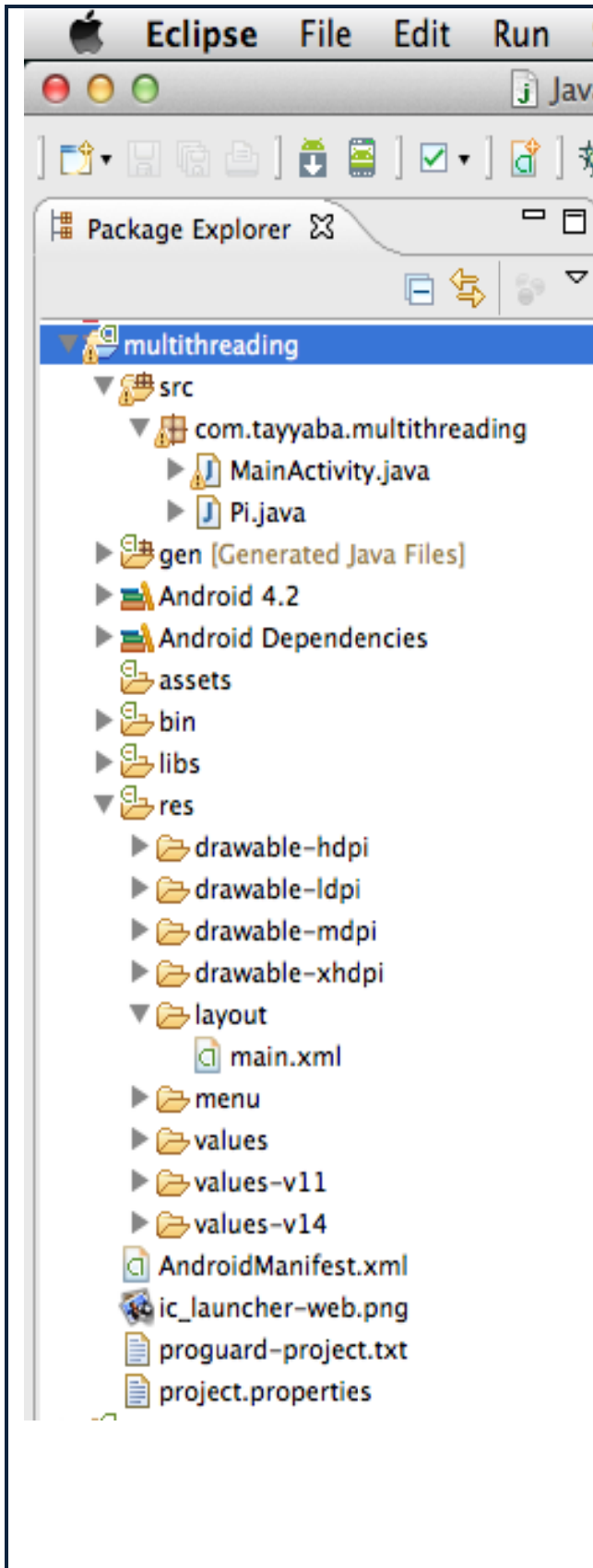
File creator Android application layer	POISX conformance file creatorAndroid example
	<pre> #define _POSIX_SOURCE 1 #include<stdio.h> int main(){ FILE *fp; char ch; int c; fp=fopen("data.txt", "w"); printf("\nTHIS DATA WRITTEN TO A FILE: "); while((ch=getchar()) !=EOF) putc(ch, fp); fclose(fp); fp=fopen("data.txt", "r"); c = fgetc(fp) ; while (c!= EOF) { putchar(c); c = fgetc(fp); printf("\nTHIS DATA WRITTEN TO A FILE: "+c); }fclose(fp);} </pre>

Table 3.4:File creator application framework

Multithreading Android application layer	POISX conformance Multithreading Android example
--	--



```

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#define NTHREADS 5
struct thread_data {
    int my_id;
    pthread_t tid;
    int status; };
struct thread_data
thread_data[NTHREADS];
void *Thread(void *);
void Error(const char
*str, ...);
void *
Thread(void *v)
{
    struct thread_data *p
= (struct thread_data *)v;
    int n;

    printf("start thread
%d\n", p->my_id);
    for (n = 0; n
<1000000000; ++n)
        ;
    p->status = p->my_id
+ 1;
    printf("end thread
%d\n", p->my_id);
    return v; }
int
main(int argc, char
**argv)
{int e, n;
    pthread_attr_t attr;
    struct thread_data
*res;
    if
(pthread_attr_t(&attr)
)
        Error("pthread_attr_i
nit");
    for (n = 0; n <NTHREADS;
++n) {
        thread_data[n].my_id
= n;
        if (e =
pthread_create(&thread_da
ta[n].tid, &attr, Thread,

```

	<pre> (void *)&thread_data[n])) Error("pthread_create %d", e);} for (n = 0; n <NTHREADS; ++n) {if (pthread_join(thread_data [n].tid, (void **)&res)) Error("pthread_join") ; printf("das war thread %d, status %d\n", res- >my_id, res- >status);}return0;} void Error(constchar *str, ...) {va_list ap; int e; e = errno; va_start(ap, str); vfprintf(stderr, str, ap); va_end(ap); fprintf(stderr, "\n"); if (e) { errno = e; perror("system error"); fprintf(stderr, "errno %d\n", errno); }exit(1);} </pre>
--	---

Table 3.5: Multithreading application framework

This step includes the conversion of the sample-tested example Android application framework into POSIX.1 template. Using the C library POSIX threads. [29] This is point-to-point conversion like headers of Android replace with according to standard C into system header or local headers, variable into local variables etc.

3.3. Android Application template for POSIX.1

POSIX.1 template is not stringent. It would be change according to the Application or need of the developer. But the #define _POSIX_SOURCE 1 is compulsory part of any application. [20]

Template	Description
/* Feature test switches */ #define _POSIX_SOURCE 1	define the <code>_POSIX_SOURCE</code> macro to enable the POSIX symbols and disable all unspecified symbols.
/* System headers */	Each Standard C or POSIX function has one or more headers that must be included to define the symbols used by that function.
/* Local headers */	Most projects have at least one project header. These define common data structures and symbols that are used in many files.
/* Macros */	Define all of your macros here.
/* File scope variables */	These are variable that are shared by several functions in the same file.
/* External variables */	This is the list of variables defined in other modules and used in this module.
/* External functions */	There should be a prototype for each user-written external function that you use.
/* Structures and unions */	Define all of the structures that are used only in this file. Any structure that is used in multiple files should be in a local header file.
/* Signal catching functions */	Place signal catching functions in one place. Signals are an unusual calling mechanism and often hard to debug. Unless you point it out clearly in your source code, it may not be obvious that something is a signal catching function.
/* Functions */	Define functions here.
/* Main */	There is a <code>main()</code> function in this file

Table 3.6: Android POSIX.1 application template

3.3.1. Used some core Portable functions

The `fgetc()`, `getc()` and `getchar()` Functions are very portable. For example in file creation, deletion and read data from it .the application used these functions for reading data from created file.

```
c = fgetc(fp);  
while (c != EOF)  
{  
    putchar(c);  
    c = fgetc(fp);  
}
```

The call `fgetc(stream)` returns the next character from stream. If stream is at end-of-file, EOF is returned. The `getc()` function is the same as `fgetc()` except it may be implemented as a macro. These functions are very portable. So through these portable functions we are able to write a portable calls like for reading a data from text file

```
char *fgets(char *s, int n, FILE *stream);
```

3.3.2. Opening and Closing Files functions

The `fopen()` function is used to connect a file with a stream:

```
fp=fopen("data.txt","w");
```

- Create text file with name data and
- w Create new file for writing. If a file with this name already exists, its contents are lost.

Some systems make a distinction between text files and binary files. While there is no such distinction in POSIX, a 'b' may be appended to the mode string to indicate binary. The b does not do anything but may be useful for compatibility with non-POSIX systems. If you are creating a binary file, include the b to make your program more portable. Most systems that do not support the b option will just ignore it.

Upon success, the fopen() function returns a pointer to a file descriptor. This pointer is used only as an argument to other functions. Do not attempt to manipulate the object it points at. If the open fails, fopen() returns a null pointer.

When you are finished with a file, you should close it. The call fclose(stream) will complete any pending processing, release system resources, and end access to the file. If there are no errors, fclose() returns zero. It returns EOF if any errors are detected.

```
int main(){
FILE *fp;
char ch;
int c;
    fp=fopen("data.txt","w");
printf("\nTHIS DATA WRITTEN TO A FILE:");
while((ch=getchar())!=EOF)
putc(ch,fp);
fclose(fp);
```

Figure.3.1 Text File creations, opening and closing code of sample Android POSIX.1 application

3.4. Sample examples code matching with Android Application template for POSIX.1

Template	Hello world Example
/* Feature test switches */ #define _POSIX_SOURCE 1	#define _POSIX_SOURCE 1
/* System headers */	#include <stdio.h>

/* Main */	<pre>int main(int argc, constchar * argv[]) { printf("%s", "Hello, World!\n"); return0; }</pre>
Template	Text file creator example
/* Feature test switches */ #define _POSIX_SOURCE 1	#define _POSIX_SOURCE 1
/* System headers */	#include<stdio.h>
/* Main */ /* Functions */	<pre>int main(){ fp=fopen("data.txt", "w"); printf("\nTHIS DATA WRITTEN TO A FILE: "); while((ch=getchar())!=EOF) putc(ch, fp); fclose(fp); fp=fopen("data.txt", "r"); c = fgetc(fp) ; while (c!= EOF) { putchar(c); c = fgetc(fp); printf("\nTHIS DATA WRITTEN TO A FILE: "+c); } fclose(fp); }</pre>
/* File scope variables */	<pre>FILE *fp; char ch; int c;</pre>
/* External functions */	<pre>fclose(fp); putchar(c); fopen("data.txt", "r");</pre>
Template	Timer example
/* Feature test switches */ #define _POSIX_SOURCE 1	#define _POSIX_SOURCE 1
/* System headers */	<pre>#include <stdlib.h> #include <stdio.h> #include <time.h></pre>

<pre>/* Main */ /* External functions */</pre>	<pre>main (argc, argv) { struct tm *tmptr; timer = time(NULL); tmptr = localtime(&timer); printf("The current time is: \n%s", ctime(&timer)); if (tmptr ->tm_isdst) printf("Daylight savings time\n"); else printf("Standard time\n"); exit(EXIT_SUCCESS); } </pre>
<pre>/* File scope variables */</pre>	<pre>int argc; char **argv;</pre>
<pre>/* Structures and unions */</pre>	<pre>struct tm *tmptr; time_t timer;</pre>
Template	Multithreading example
<pre>/* Feature test switches */ #define _POSIX_SOURCE 1</pre>	<pre>#define _POSIX_SOURCE 1</pre>
<pre>/* System headers */</pre>	<pre>#include <pthread.h> #include <stdio.h> #include <stdarg.h> #include <errno.h> #include <stdlib.h> #include <unistd.h> #define NTHREADS 5</pre>
<pre>/* Structures and unions */</pre>	<pre>struct thread_data { int my_id; pthread_t tid; int status; }; struct thread_data thread_data[NTHREADS]; void *Thread(void *); void Error(constchar *str, ...); void * Thread(void *v) { struct thread_data *p = (struct thread_data *)v; int n; printf("start thread %d\n", p- >my_id); </pre>

	<pre> for (n = 0; n <1000000000; ++n) ; p->status = p->my_id + 1; printf("end thread %d\n", p- >my_id); return v; } </pre>
<pre> /* Main */ /* External variables */ /* External functions */ </pre>	<pre> int main(int argc, char **argv) { int e, n; pthread_attr_t attr; structthread_data *res; if (pthread_attr_init(&attr)) Error("pthread_attr_init"); for (n = 0; n <NTHREADS; ++n) { thread_data[n].my_id = n; if (e = pthread_create(&thread_data[n].tid, &attr, Thread, (void *)&thread_data[n])) Error("pthread_create %d", e); } for (n = 0; n <NTHREADS; ++n) { if (pthread_join(thread_data[n].tid, (void **)&res)) Error("pthread_join"); printf("das war thread %d, status %d\n", res->my_id, res- >status); }return0; } </pre>
<pre> /* Signal catching functions */ </pre>	<pre> void Error(constchar *str, ...) { va_list ap; int e; e = errno; va_start(ap, str); vfprintf(stderr, str, ap); va_end(ap); fprintf(stderr, "\n"); if (e) { errno = e; perror("system error"); fprintf(stderr, "errno %d\n", </pre>

	<pre> errno); } exit(1); } </pre>
--	-----------------------------------

Table 3.7: matching of POSIX.1 compliant Android application with POSIX.1 template

3.5. Tested Sample Examples

The sample examples are running in two different environments. Before applying the POSIX .1 thin layer model. These applications run in to the Eclipse INDIGO version with Android virtual device (AVD) Intel Atom x86 with API level 10.

After Applying the POSIX .1 thin layer model these are tested in to Xcode with gcc compiler version is mention in previous Fig.9 at Macintosh OS. So results of the testing applications are very adorable. Resultant applications are compatible and reusable.

Sample Example Applications Comparison Table	
Text File creator POSIX Conformance Android application	File creator Android application
<pre> #define _POSIX_SOURCE 1 #include<stdio.h> int main(){ FILE *fp; char ch; int c; fp=fopen("data.txt", "w"); printf("\nTHIS DATA WRITTEN TO A FILE: "); while((ch=getchar())!=EOF) putc(ch, fp); fclose(fp); //char *fgets(char *s, int n, FILE *stream); fp=fopen("data.txt", "r"); c = fgetc(fp) ; while (c!= EOF) { putchar(c); c = fgetc(fp); </pre>	<pre> package com.tayyaba; import java.io.BufferedReader; import java.io.FileNotFoundException; import java.io.IOException; import java.io.InputStream; import java.io.InputStreamReader; import java.io.OutputStreamWriter; import android.app.Activity; import android.content.Context; import android.os.Bundle; import android.util.Log; import android.view.View; import android.widget.EditText; import android.widget.TextView; import android.widget.Toast; public class FileActivity extends Activity {private static final String TAG = FileActivity.class.getName(); </pre>

```

printf("\nTHIS DATA WRITTEN TO A
FILE: "+c);
}

fclose(fp);
}

```

```

private static final String
FILENAME = "myFileTayyaba.txt";

@Override
public void onCreate(Bundle
savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);}
    public void SaveText(View view){
        // EditText ET =
        (EditText)findViewById(R.id.editText1);
        EditText ET =
        (EditText)findViewById(R.id.editText1);
        String
textToSaveString =
ET.getText().toString()
        //String
textToSaveString = "Hello Android
tayyaba";
        writeFile(textToSaveString);
        String
textFromFileString = readFromFile();
        if (
textToSaveString.equals(textFromFileStri
ng) )
            Toast.makeText(getApplicationCo
ntext(), "both string are equal",
Toast.LENGTH_SHORT).show();
        else
            Toast.makeText(getApplicationCo
ntext(), "there is a problem",
Toast.LENGTH_SHORT).show();

        Toast.makeText(this,"Text Saved
!",Toast.LENGTH_LONG).show();}
    private void writeFile(String
data) {
        try {
            OutputStreamWriter
outputStreamWriter = new
OutputStreamWriter(openFileOutput(FIL
ENAME, Context.MODE_PRIVATE));
            outputStreamWriter.write(data);
            Log.e(TAG, "File write : ");
            outputStreamWriter.close();}
        catch (IOException e) {
            Log.e(TAG, "File write failed: " +

```

```

e.toString());} }
    private String readFromFile() {
        String ret = "";
        try {
            InputStream inputStream =
openFileInput(FILENAME);
            if ( inputStream != null ) {
                InputStreamReader
inputStreamReader = new
InputStreamReader(inputStream);
                BufferedReader bufferedReader =
new BufferedReader(inputStreamReader);
                String receiveString = "";
                StringBuilder stringBuilder = new
StringBuilder();

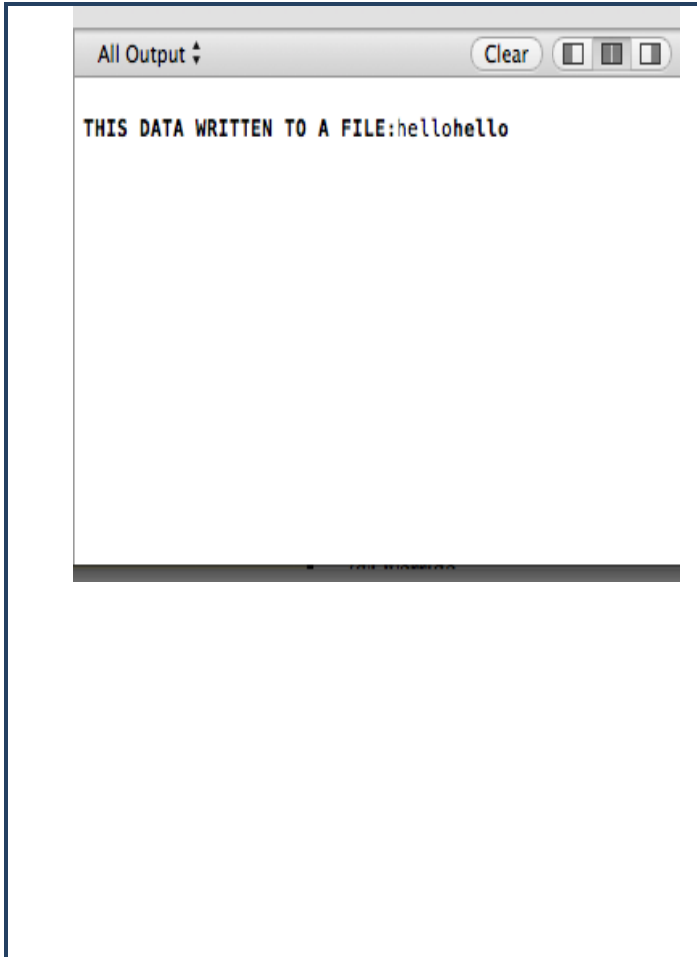
                while ( (receiveString =
bufferedReader.readLine()) != null ) {

                    stringBuilder.append(receiveStrin
g);}inputStream.close();
                    ret = stringBuilder.toString();
                    TextView tv =
(TextView)findViewById(R.id.textView1
);
                    tv.setText("text file data:"+ret);
                    Log.e(TAG, "Can read file: " +
ret.toString());
                }
            }
            catch (FileNotFoundException e) {
                Log.e(TAG, "File not found: " +
e.toString());
            }
            catch (IOException e) {
                Log.e(TAG, "Can
not read file: " + e.toString());
            }

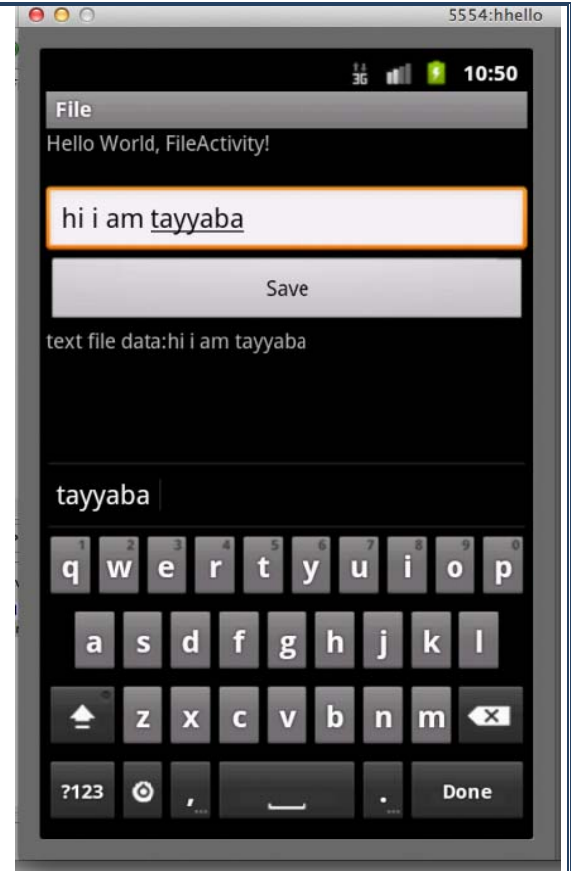
            return ret;
        }
    }
}

```

Output



**Standard time display POSIX Conformance
Android Application**



**Standard time display Android
Application**

```

#define _POSIXSOURCE 1
#include <stdio.h>
#include <time.h>
main(argc,argv)
int argc;
char **argv;
{
    struct tm *tmptr;
    time_t timer;
    timer = time(NULL);
    tmptr = localtime(&timer);
    printf("The current time is:\n%s",
           ctime(&timer));
    if (tmptr -> tm_isdst)
        printf("Daylight savings time\n");
    else
        printf("Standard time\n");
    exit(0);
}

```

```

public class DateActivity extends
Activity {
    /** Called when the activity is first
    created. */
    @Override
    public void onCreate(Bundle
    savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.main);

        Calendar c = Calendar.getInstance();
        System.out.println("The current time
        is: "+c.getTime());
        SimpleDateFormat df = new
        SimpleDateFormat("EEE LLL dd
        HH:mm:ss");
        String formattedDate =
        df.format(c.getTime());
        // Now we display formattedDate
        value in TextView
        TextView txtView = new
        TextView(this);
        txtView.setText("The current time is:
        "+formattedDate+" "+"Standard time");

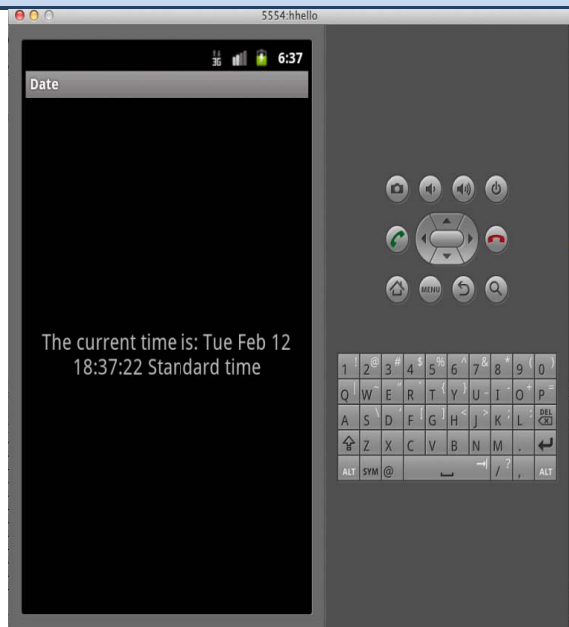
        txtView.setGravity(Gravity.CENTER);
        txtView.setTextSize(20);
        setContentView(txtView);
    }
}

```

OUTPUT

All Output

```
The current time is:
Tue Feb 12 12:47:20 2013
Standard time
```



**Multithreading POSIX Conformance
Android Application**

Android Multithreading application

```
#define _POSIX_SOURCE 1
#include <pthread.h>
#include <stdio.h>
#include <stdarg.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>

#define NTHREADS 5

struct thread_data {
    int my_id;
    pthread_t tid;
    int status;
};

struct thread_data
thread_data[NTHREADS];

void *Thread(void *);
void Error(constchar *str, ...);

void *
Thread(void *v)
{
    struct thread_data *p =
```

```
package
com.tayyaba.multithreading;

import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.app.Activity;
import android.util.Log;
import android.view.Menu;
import
android.widget.TextView;
publicclass MainActivity
extends Activity {
    TextView txt;
    // our handler
    Handler handler =
newHandler() {
    @Override
    publicvoid
handleMessage(Message msg) {
    // get the bundle and
extract data by key
    Bundle b =
```

```

(structthread_data *)v;
    int n;

    printf("start thread %d\n",
p->my_id);
    for (n = 0; n <1000000000;
++n)
        ;
        p->status = p->my_id + 1;
        printf("end thread %d\n", p-
>my_id);

    return v;
}

int
main(int argc, char **argv)
{
    int e, n;
    pthread_attr_t attr;
    structthread_data *res;

    if
(pthread_attr_init(&attr))
        Error("pthread_attr_init");

    /*
        if ((e =
pthread_attr_setscope(&attr,
PTHREAD_SCOPE_SYSTEM))

Error("pthread_attr_setscope
%d", e);
        */

    for (n = 0; n <NTHREADS;
++n) {
        thread_data[n].my_id = n;
        if (e =
pthread_create(&thread_data[n].t
id, &attr, Thread,
(void *)&thread_data[n]))
            Error("pthread_create
%d", e);
    }

    for (n = 0; n <NTHREADS;
++n) {
        if
(pthread_join(thread_data[n].tid
, (void **)&res))
            Error("pthread_join");

```

```

msg.getData();
        String key =
b.getString("My Key");
        txt.setText(txt.getText()
+ "new " + key

+System.getProperty("line.sepa
rator"));
    }
    };

    /** Called when the
activity is first created. */
    @Override
    publicvoid
onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInsta
nceState);

        setContentView(R.layout.main);
        txt = (TextView)
findViewById(R.id.txt);
    }

    @Override
    protectedvoid onStart() {
        super.onStart();
        super.onStop();
        // create a new thread
        booleanbackground = true;
        Thread background1 =
new Thread(new Runnable() {

        @Override
        publicvoid run() {
            for (int i = 0; i < 10;
i++) {
                try {
                    Thread.sleep(1000);

Message msg = new Message();

Bundle b = new Bundle();

        b.putString("My Key", "My

```

```

        printf("das war thread %d,
status %d\n",
        res->my_id, res-
>status);
    }
    return 0;
}

void
Error(constchar *str, ...)
{
    va_list ap;
    int e;

    e = errno;
    va_start(ap, str);
    fprintf(stderr, str, ap);
    va_end(ap);
    fprintf( stderr, "\n" );
    if (e) {
        errno = e;
        perror("system error");
        fprintf( stderr, "errno
%d\n", errno);
    }

    exit(1);
}

```

```

thread#: " +
String.valueOf(i));
        msg.setData(b);

        handler.sendMessage(msg);
    }
    catch (Exception e)
    {
        Log.v("Error",
e.toString());
    }
}

});

background1.start();

background1.interrupt();

System.out.println("thread
kill");

}
}

```

Output

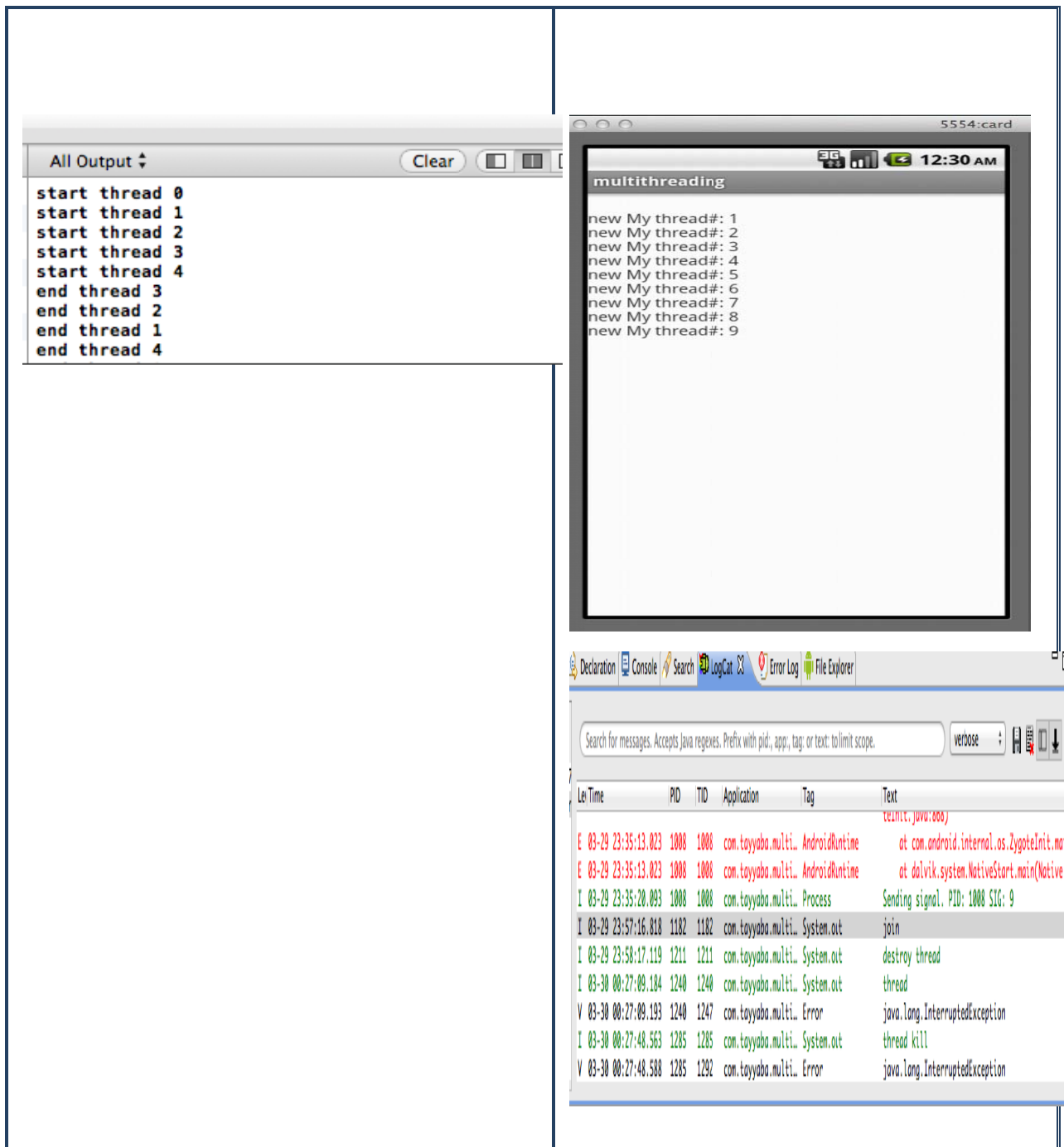


Table 3.8: Tested sample Android applications code and output comparison

3.6. Chapter summary

In this chapter we have briefly explained the actual implementation of ANDROID POSIX.1 Thin layer Model. The information about the development environment is given and all sample examples Android applications layers are explained there. After that template is described for Android application for converting in to POISX. Then sample examples code match with this template. At the end sample example code snippets are presented. Analysis and comparison of the sample examples code also explained.

Chapter 5

4. Analysis and Results

Essentially we are trying to provide the standardization (through POSIX) and portability of Android applications on multiple operating systems. Android Application developers have the opportunity to establish a template that assists in producing code with consistent format. Because a well-structured program is portable among the different programmers who may maintain it. Placing program elements in a consistent order makes finding things easier. [17] Consequently the concluded points of this experiment are:

- **Portability**
 - POSIX .1 thin layer model is initiation point for Android application portability to different operating systems.
- **Reusability**
 - POSIX .1 thin layer model provides the reusability of the Android applications on multiple operating systems.
- **Standardization**
 - POSIX .1 thin layer model is a standard way of transformation of application with damaging the application internal structure.
- **Diversity**
 - POSIX .1 thin layer model gives the diversity to the Android application market.

4.1. Functioning Outside the Standards

Most programs have only a few areas, which need to go outside of the standards. Keep those area isolated to a few modules. Keep most of the code POSIX conforming. For example, I have a File creation, deletion and read file data Android application, which create file, read its data and delete it. The program's structure is shown in Figure 11.

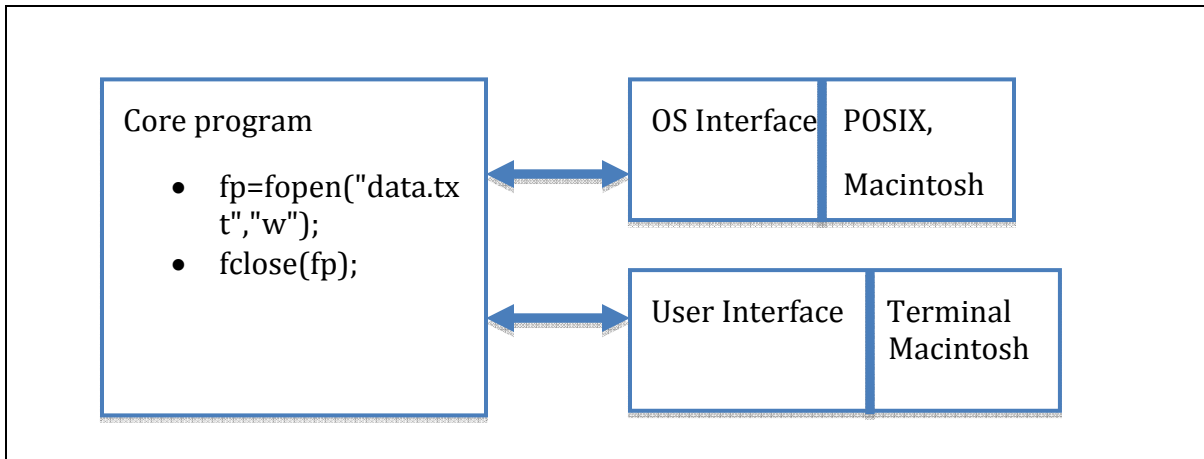


Figure.4.1 Example of portable application

Modules in the Program Core do not have any knowledge of the user interface. If I need, for Example, to get data from the user to write on file, I call `fopen("data.txt","w");`. That is one of the routines I wrote in the user interface module. On a system with a graphic user interface, like Android there is text box for taking data from user and the system with out graphic user interface there is a terminal used for taking the data from the user. But the core functionality of the programs is same

Most of the program remains unchanged over several operating systems and user interfaces. I can build a version for different operating systems and user interfaces by changing that module.

4.2. Quantified Feasibility analysis

At this time Android covers the 53% of the Smartphone market share as shown in figure in 12. [27]. But we turn into 100% by introducing POSIX.1 Thin layer Model. It provides the viability for Android mobile users as well as developers. After implementation of this POSIX.1 Thin layer Model the Android applications can run on any operating system so the Android covers the 100% market, which means the revenue according to figure 12 it would be double. The statistical result is given in the table.12. This model focuses on the Android mobile users and Android developer through reusability and standardization

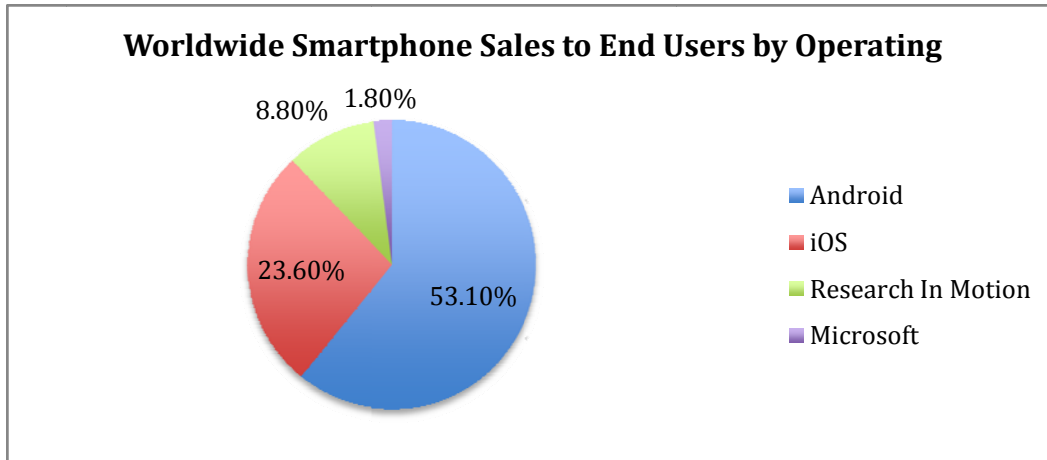


Figure.4.2 Worldwide Smartphone Sales to End User by Operating source: Gartner (February 2013)[23]

Nowadays Android developers because of open source SDK and API support prioritize an Android market. [23] The Android applications market revenue increasing very rapidly [21]. Like in figure.1 Android market growth is 861.5%, which is tremendous. But question is here Why are restricting Android market only to the Mobile OS.

Global Mobile Applications Store Ranking in 2010 and 2009
(Ranking by Revenue in Millions of U.S. Dollars)

2010 Rank	Store	2009 Revenue	2009 Share	2010 Revenue	2010 Share	Year-Over-Year Growth
1	Apple App Store	\$769	92.8%	\$1,782	82.7%	131.9%
2	BlackBerry App World	\$36	4.3%	\$165	7.7%	360.3%
3	Noka Ovi Store	\$13	1.5%	\$105	4.9%	719.4%
4	Google Android Market	\$11	1.3%	\$102	4.7%	861.5%
	Total	\$828	100.0%	\$2,155	100.0%	160.2%

Source: IHS Screen Digest February 2011

Figure.4.3 Android market share [21]

The market of the desktop OS is very large as shown in the below table.12.but if we merge both these markets only for the Android the result is very magnificent in the form of revue however also in the form of manpower reduction which shown in the table.13.

POSIX Compliant operating systems	Market share of desktop Operating system	Android market share	Total market share for developers
Windows 7	44.55%	53.1%	97.65%
Windows XP	38.99%	53.1%	92.09%
Windows Vista	5.17%	53.1%	58.27%
Mac OS X 10.8	2.61%	53.1%	55.71%
Windows 8	2.67%	53.1%	55.77%
Mac OS X 10.6	1.97%	53.1%	55.07%
Mac OS X 10.7	1.93%	53.1%	55.03%

Table 4.1: Market share analysis for Android developers [6]

The above table gives the brief assessment for developer after usage of Android POISX.1 thin layer model. It clearly shown that POISX Desktop OS users and mobile OS users covered very impressive percentage of market and after combing these target markets the revenue of the developer is twice.

Application type	Application development manpower	Application development time	POSIX application development time	POSIX application development manpower
Entertainment	6-7 developers	30 days (min) 120 days (max)	3 developers	15 days (min) 60 days (max)

Lifestyle	2-3 developers	20 days (min)	1 developer	10 days (min)
		60 days (max)		30 days (max)
Productivity	10-11 developers	30 days (min)	5-6 developers	15 days (min)
		120 days (max)		60 days (max)
Libraries & Demo	10-11 developers	30 days (min)	5-6 developers	15 days (min)
		120 days(max)		60 days (max)

Table 4.2: Comparative analysis of applications development time and manpower for Android developers

Android POISX.1 thin layer model also provides the reusability concept for developer in reducing the manpower since usage of the core functional modules. These core functional modules use by developer again and again in multiples Android applications. Secondly developer developed the application at once according to the POISX.1 template and its will be available for the mobile OS user as well as for Desktop OS users. The above table had shownextraordinarily results that the manpower would be half after implementation of this model.

4.2.1. Resulting impact factor for Android developers

The feasibility study of the POSIX.1 thin layer Model clearly revealed a lot of benefits for developers. The resultantinfluencing factors are shown in below figure 14

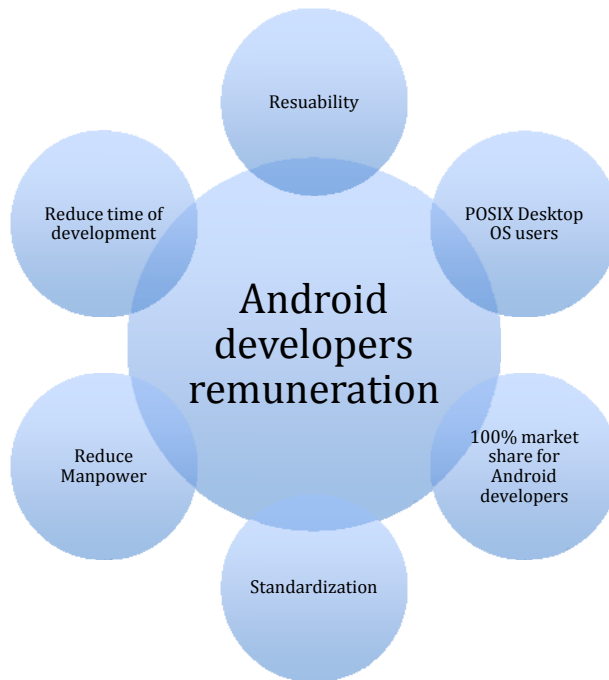


Figure.4.4 POSIX.1 thin layer Model resulting impact factors

4.3. Chapter summary

In this chapter we narrated the different statistical and dynamic factor of this POSIX.1 Thin layer Model .we evaluated perform of this model from different point of views like users and developers. The resulting impact factors are shown in the form of tables. These results are rough estimates, which may be varies with respect of usage.

Chapter 6

5. Conclusion

Right now, Android is the most famous operating system out of the numerous Linux supported mobile operating systems (e.g., Maemo) [22]. POSIX .1 thin layer model assigns the Android applications to a wider marketplace without restricting to them with only mobile computing. In this work, the main theme of research provides the portability to the Android Application with POSIX.1 standard. In summary introducing this thin layer POSIX.1 model expands the market for Android applications as well as adds real-time capability and higher reusability. This meticulous work provides the new horizon for developers in form of increasing revenue as well as reduces the manpower. Basically we are trying to vanish the dependencies of Android applications by providing the standard approach.

5.1. Achievement

This layer resolved compatibility challenges for Android applications .Now Android application would be accessible for any Android mobile OS in addition to POISX desktop OS.

Android applications for all POSIX Desktop OS. It means one Android application market is serviceable for mobile OS user along with Desktop OS users.

According to the research point of view Android POISX.1 thin layer model award the mobile computing new horizon of research. This research lead to solve the real time mobile computing problem like portability, reusability and transformation, secondly it increase the Android market revenue in addition to target market enrichment by adding the POISX desktop users.

The one research paper related to this research is accepted in IOSR Journal of Computer Engineering (IOSR-JCE).

We got another acceptance notification from the 2013 9th International Conference on Natural Computation (ICNC 2013), to be held jointly with the 2013 10th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2013), from 23-25 July 2013 in Shenyang, China (impact factor 1.6).

5.2. Limitations

We have faced multiple limitations related to POSIX as well as related to the Android applications.

- POSIX have a list of standards and some of these standards are not still verified with IEEE. Secondly POISX bonding languages are very extinct so POISX programming is very difficult tasks.
- With passing each day POSIX standards are modified very frequently. These abrupt changes in standards becomes the developer life miserable.
- A lot of Android applications are GUI dependent and POISX .1 support very limited GUI features so need of GUI functions in POISX .1.
- There is no standard tool or engine for langue conversion from java to standard C.

5.3. Future work

Till now there is only Application layer (code) implementation through this model but the need of implementation of UI layer is very stimulating and tempting. Although XML code conversion is very difficult and C library limitation for interface. [30] The Hardware acceleration for Graphics subsystem is also in require for completing implementation.

Second option is related to making the Android OS POSIXcompliant. This is not an easy task as there are a lot of limitations of Android hardware restriction, Android devices variety plus DVM usage that caused the performance drawback [26] but the proposed

model is one of best solution for all these limitations. The small size of usage hardware is obvious but implementation is not impossible by executing the Standard C library instead of using C/C++ for code conversion. [28] This is only the idea but achievement needs more attention. The very rough model is presented below figure.

Figure. 5.1 proposed Android OS POSIX compliant model

In this model we are try to introduce a new layer, which called the Java POISX APIs layer. This layer simply converts the all java APIs into POISX APIs but still in java language .so DVM consider it as java command and convert it into dex. Format.

At the runtime Dalvik virtual machine and the Java core libraries are part of the Android. The Dalvik virtual machine is an interpreter for byte code that has been converted from Java byte code to Dalvik byte code. [24] Dalvik itself is compiled to native code whereas the core libraries are written in Java, thus interpreted by Dalvik. It means conversion of java to C/C++ done here. But we are try to convert the Android OS POISX .One solution

is the POSIX package. This package provides access to the POSIX API from Java. However essential question is that where put this POISX APIs library for Java?

As shown in above figure .15 we put the java POSIX APIs layer that Basically DVM do the conversion of java applications to .dex format means conversion of java to C/C++ .so DVM has not problem if there is any JAVA API so if we put the JAVA POSIX API [25] layer which convert the all java simple APIs to POISX APIs but still in the java language. So DVM very easily do it conversion because DVM consider it a java API.

References

Chapter 1	
[1]	Google Android - An Open Handset Alliance Project, 2008. http://code.google.com/android/
[2]	Hassan Reza, and Narayana Mazumder, “A Secure Software Architecture for Mobile Computing”, (2012 IEEE)
[3]	Donald A. Lewine, “POSIX Programmer's Guide Writing Portable UNIX Programs with the POSIX.1 Standard”, 1991,pp.31-36
[4]	IEEE/ANSI Std 1003.1: Information Technology-- (POSIX®)--Part 1: System Application: Program Interface (API) [C Language], includes (1003.1a, 1003.1b, and 1003.1c). 1996.
[5]	What is Android, Http://www.android.com/about/ . Accessed March 2013
[6]	Market share report of Android, Http://www.netmarketshare.com .Accessed March 2013
Chapter 2	
[7]	Introduction to Android development, Http://www.ibm.com/developerworks/opensource/library/os-android-devel/ . Accessed March 2013
[8]	Android operating system, Http://en.wikipedia.org/wiki/Android_(operating_system) . Accessed December

	2012
[9]	Kolin Paul, Tapas Kumar Kundu "Android on Mobile Devices: An Energy Perspective," 10th IEEE International Conference on Computer and Information Technology, 2010.
[10]	Android, the world's most popular mobile platform, Http://developer.android.com/about/index.html .Accessed December 2012
[11]	Android market analysis, Http://en.wikipedia.org/wiki/Android_Market .Accessed December 2012
[12]	SDK for Android developers, Http://developer.android.com/sdk/eclipse-adt.html .Accessed December 2012
[13]	Introduction of Android, Http://www.android.com/about/ . Accessed December 2012
[14]	Bill O. Gallmeister, "POSIX. 4: Programming for the Real World", 1995.ppt.4, 19-20,22,23
Chapter 3	
[15]	App framework, http://developer.android.com/about/versions/index.html . Accessed December 2012
[16]	Kyosuke Nagata, Saneyasu Yamaguchi "An Android Application Launch Analyzing System", (2012,IEEE)
[17]	Donald A. Lewine, "POSIX Programmer's Guide Writing Portable UNIX Programs with the POSIX.1 Standard", 1991,pp.16-17, 25
[18]	POSIX. 1: ISO/IEC 9945-1:1990 IEEE Std. 1003.1-1990

[19]	IEEE Std 1003.1b-1993 (Formerly known as IEEE P1003.4) (Includes IEEE Std 1003.1-1990)
Chapter 4	
[20]	IEEE Portable Applications Standards Committee, P1003.13: Information Technology - Standardized Applications Environment Profile - POSIX Real-time Application Support (AEP) (Draft 5) (Feb 1992).
[21]	Apple's rivals battle for iOS scraps as app market sales grow to \$2.2 billion, Http://appleinsider.com/articles/11/02/18/rim_nokia_and_googles_android_battle_for_apples_ios_scraps_as_app_market_sales_grow_to_2_2_billion.html . Accessed March 2012
[27]	Android Market news, Http://www.gartner.com/newsroom/id/2335616 . Accessed March 2012
[29]	Frank Mueller, "A library implementation of POSIX threads under UNIX", Proceedings of the USENIS Conference (Jan 1993) 29-41.
Chapter 5	
[23]	Chung-Shih, Yi-Kai, Chin-Yuen, Ying-Dian, and Gong-Da, "An Innovative ICT Service Creation Approach based on IMS and Android Collaboration". (2009, IEEE)
Chapter 6	
[22]	E. Oliver, "A Survey of Platforms for Mobile Networks Research. Mobile Computing and Communications Review", December 2008, pp. 56-63.
[24]	Namseung Lee, Sung-Soo Lim, "A Whole Layer Performance Analysis Method

	for Android Platforms”, (2011 IEEE).
[25]	Java POSIX APIs, Http://bmsi.com/java/posix/posix-1.2.2/doc/index.html . Accessed March 2012
[26]	Leonid, Aubrey-Derrick, Hans-Gunther, Ahmet Camtepe and Sahin Albayrak,” Developing and Benchmarking Native Linux Applications on Android,” Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Volume 7, pp 381-392, 2009.
[28]	E. Cooper and R. Draves, “C threads”. TR CMU-CS-88- 154, Carnegie Mellon University, Dept. of CS (1988).
[30]	J. B. Fenwick, Jr., B. L. Kurtz, and J. Hollingsworth, “Teaching mobile computing and developing software to support computer science education,” in Proceedings of the 42nd ACM technical symposium on Computer science education, ser. SIGCSE ’11. New York, NY, USA: ACM, 2011, pp. 589–594. [Online]. Available: http://doi.acm.org/10.1145/1953163.1953327

