POSIX.1 conformance for Android Applications

TayyabaNafees
Department of computer engineering
National University of science &Technology, H-12,
Islamabad, Pakistan

Prof. Dr. Shoab Ahmad Khan
Department of computer engineering
National University of science &Technology, H-12,
Islamabad, Pakistan

Abstract—**Android operating system is designed for use in mobile computing by The Open Handset Alliance. 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 POISX Desktop operating system without disturbing the internal structure of application is very desirable. 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 POISX desktop 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 POISX Desktop operating systems and Android applications. We have analyzed our 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% plus add real-time standardization and reusability.**

Keywords—Portable Operating System Interface (POSIX); Application Programming Interface (API); Operating system (OS); User interface (UI)

I. Introduction

Android is open source mobile OS .It is particularly adapted by various manufacturers and modified based on their own taste for its openness [1]. Currently, Android cell phones are becoming more sophisticated by providing functionalities that once expected from laptop and/or desktop computing systems. [2] For example, using cell phone, callers can now interact with system using spoken language, brows internet, exchange emails, chat online and social network medias, use navigation systems, etc. 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 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. [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:

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

A. The Problem Statement

Android is open source OS introduce by Google. Android is still developing. In the design of today's computing systems it is becoming increasingly important to design software with an open system architecture utilizing industry adopted standards. The need to develop open systems is driven by these major factors.

- Inefficient usage of manpower: First, gone are the days where a single developer can implement the entire system from scratch. Software development programs are continuously growing in scale, requiring teams of increasing size.
- Portability problem: Secondly, software does not operate in isolation; it must co-exist with the vast amount of commercially available software and can be run on available OS.
- Maintainability problems: The lifecycle of a software application is typically long requiring numerous modifications and updates as new features are added.
- 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.

*1) Android Application Portability:*There are more than 600,000 apps and games available on Google Play store. [8] 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 looking to employ Android in a variety of other embedded systems that have traditionally relied on the benefits of true real-time operating systems performance, boot-up time, real-time response, reliability, and no hidden maintenance costs.

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.

3)Earliest Idea Invention of POSIX Compliance 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.

II. Background

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. 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.

A. What POSIX Is:

POSIX is a standard to allow applications to be source-code portable from one system to another. On a system conforming to a particular version of POSIX (as measured by the test suite it passed), you should be able to just compile and run those applications, which use the POSIX (and only the POSIX) functions. POSIX basically dependent on:

- 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 has a variety of way of compiling code, for each occurrence. [1]
- Headers: A set of headers that defines the POSIX interface supported on the particular system. [1] #include<stdio.h> was used header file in given example
- 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. [1]
- A Run-Time System: Once user has built the program, the run-time, or operating system, allows him/her to run the application. [1]

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. [1]

B. Android

Android is s a software stack for mobile devices, which includes an operating system, a middleware and key applications. Android SDK provides the tools and APIs necessary to develop an application using JAVA (which is a popular language amongst the developers. Currently, Android is the most popular operating system out of the several Linux based mobile operating systems (e.g.,Maemo) [4].

- Linux Kernel: Android is based on Linux but is not Linux. The kernel of Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. [6]
- Libraries: The surface manager of Android library takes care of the display of the system and OpenGL is an open-source utility, which takes care of graphics of the system. [6]
- Android Runtime: The development language that is used in this section is Java. The core Libraries of

Android is very powerful, yet simple and familiar development platform as it is very similar to Linux.

- Dalvik Virtual Machine (DVM): Dalvik virtual machine focuses on two of the most important issues of mobile system: limited space, and limited power. DVM converts all the files into smaller and more optimized (.dex) suitable for mobile systems [6].
- Application Framework and Application: The activity manager is responsible to keep track of life cycle of any application. [7]

III. IMPLEMENTATION OF ANDROID POSIX.1 THIN LAYER MODEL

A. *Contribution*

*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. 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.

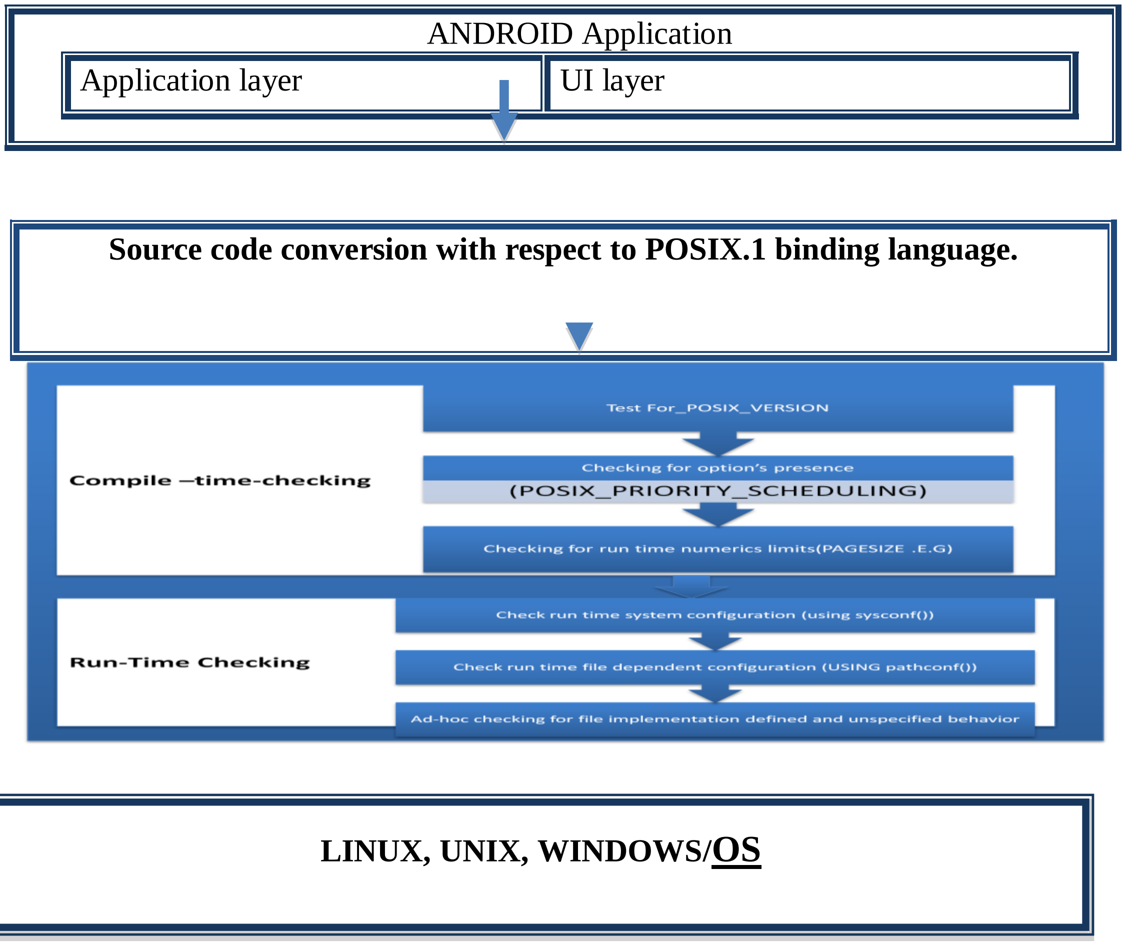

ANDROID Application
Application layer
UI layer
Source code conversion with respect to POSIX.1 binding language.
Compile –time-checking
Test For_POSIX_VERSION
Checking for option's presence
(POSIX_PRIORITY_SCHEDULING)
Checking for run time numerics limits(PAGESIZE .E.G)
Run-Time Checking
Check run time system configuration (using sysconf())
Check run time file dependent configuration (USING pathconf())
Ad-hoc checking for file implementation defined and unspecified behavior
LINUX, UNIX, WINDOWS/OS

Fig.1. Proposed POSIX.1 thin layer model

2) 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.

TABLE I. DEVELOPMENT ENVIRONMENT SPECIFICATIONS

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

3) *List of Android Applications:* For this model we start the implementation from very simple to the complex one like multithreading [29].

Hello world

- Timer
- Text file creator, save data on it and display the text on the terminal
- Multithreading example [29]

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 complier 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.

TABLE II. TIMER APPLICATION FRAMEWORK

Timer Android application layer	POISX conformance Timer Android example

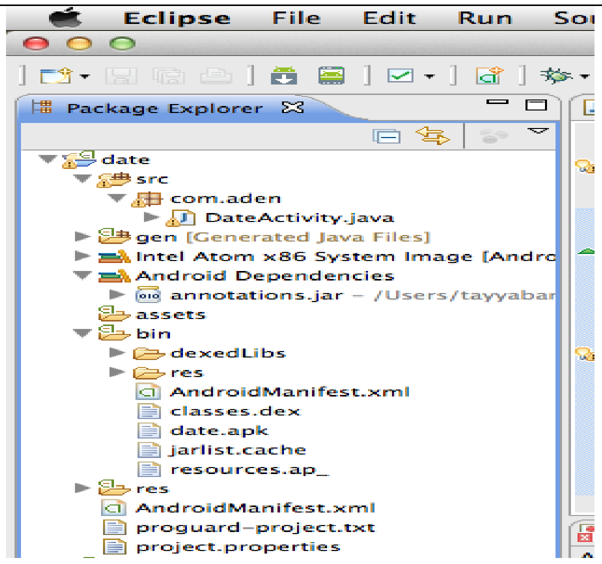
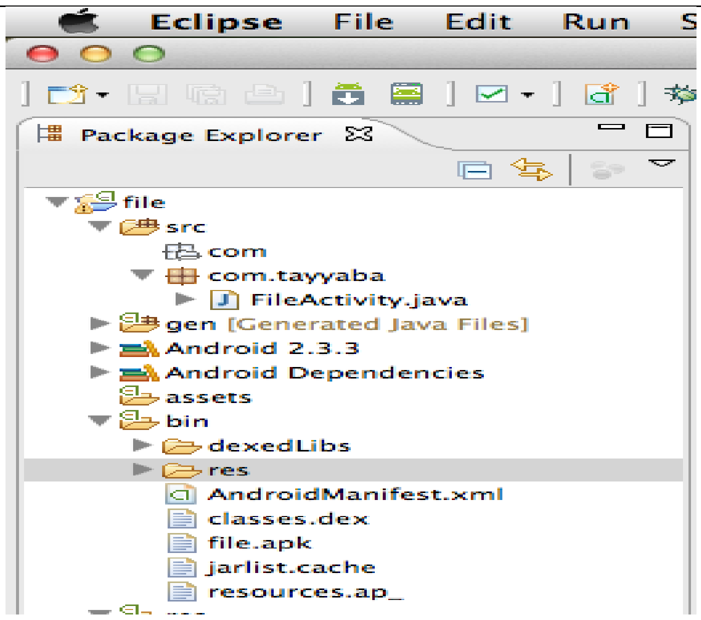
 Eclipse File Edit Run So
Package Explorer
date
src
com.aden
DateActivity.java
gen [Generated Java Files]
Intel Atom x86 System Image [Andro
Android Dependencies
annotations.jar – /Users/tayyabar
assets
bin
dexedLibs
res
AndroidManifest.xml
classes.dex
date.apk
jarlist.cache
resources.ap_
res
AndroidManifest.xml
proguard-project.txt
project.properties
	#define _POSIXSOURCE 1
	#include<stdio.h>
	#include<time.h>
	main(argc,argv)
	intargc;
	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);
	}

TABLE III. FILE CREATOR APPLICATION FRAMEWORK

File creator Android application layer	POISX conformance file creatorAndroid example

	#define _POSIX_SOURCE 1
	#include<stdio.h>
	int main(){
	FILE *fp;
	charch;
	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);}

4) *Android Application Template for POSIX.1:* This template would be change according to the Application or need of the developer. But the #define _POSIX_SOURCE 1 is compulsory part of any application. [29]

TABLE IV. ANDROID POSIX.1 APPLICATION TEMPLATE

Template	Description
/* Feature test switches */ #define _POSIX_SOURCE 1	define the _POSIX_SOURCE 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 main() function in this file

a) *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 = getc(fp) ;
while (c!= EOF)
{
putchar(c);
c = getc(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);

b) Opening and Closing File 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;
charch;
int c;
fp=fopen("data.txt","w");
printf("\nTHIS DATA WRITTEN TO A FILE:");
while((ch=getchar())!=EOF)
putc(ch,fp);
fclose(fp);
```

5) Sample examples code matching with Android Application template for POSIX.1:

TABLE V. MATCHING OF POSIX.1 COMPLIANT ANDROID APPLICATION WITH POSXI.1 TEMPLATE

Template	Text file creator example
/* Feature test switches */ #define _POSIX_SOURCE 1	#define _POSIX_SOURCE 1
/* System headers */	#include<stdio.h>

/* Main */
/* Functions */
	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);}

/* File scope variables */	FILE *fp; char ch; int c;

/* External functions */	fclose(fp);
	putchar(c);
	fopen("data.txt","r");

Template	Timer example
/* Feature test switches */ #define _POSIX_SOURCE 1	#define _POSIX SOURCE 1

/* System headers */	#include<stdlib.h>
	#include<stdio.h>
	#include<time.h>

/* Main */
/* External functions */
	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");
	elseprintf("Standard time\n");
	exit(EXIT_SUCCESS); }

/* File scope variables */	intargc; char **argv;
/* Structures and unions */	struct tm *tmptr; time_t timer;

6) Sample Examples:

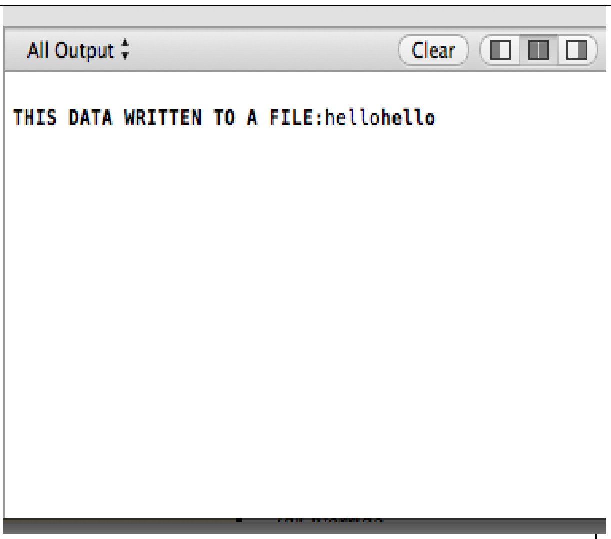
TABLE VI. TESTED SAMPLE ANDROID APPLICATIONS CODE AND OUTPUT COMPARISON

Sample Example Applications Comparison Table	
Text File creator POSIX Conformance Android application	File creator Android application
#define _POSIX_SOURCE 1	packagecom.tayyaba;

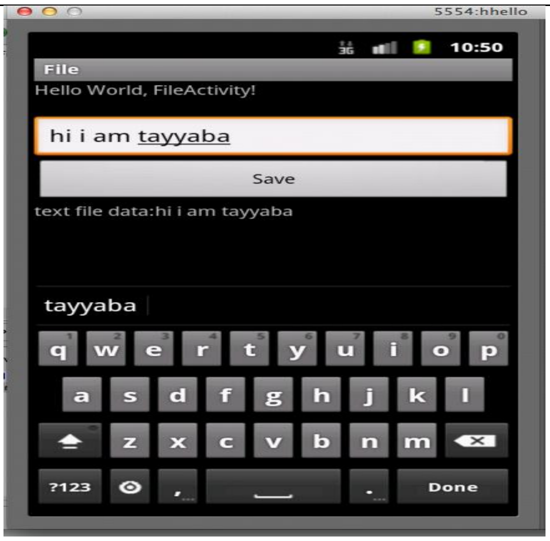
```
#include<stdio.h>
int main(){
   FILE *fp;
charch;
int c;
fp=fopen("data.txt","w");
printf("\nTHIS DATA WRITTEN TO A FILE:");
while((ch=getchar())!=EOF)
putc(ch,fp);
fclose(fp);
   //char *fgets(char *s, int n, FILE *stream);


fp=fopen("data.txt","r");
c = fgetc(fp) ;
while (c!= EOF)
   {
                    putchar(c);
                    c = fgetc(fp);
printf("\nTHIS DATA WRITTEN TO A FILE:"+c);
   }

fclose(fp);
}
```

```
importjava.io.BufferedReader;
importjava.io.FileNotFoundException;
importjava.io.IOException;
importjava.io.InputStream;
importjava.io.InputStreamReader;
importjava.io.OutputStreamWriter;
importandroid.app.Activity;
importandroid.content.Context;
importandroid.os.Bundle;
importandroid.util.Log;
importandroid.view.View;
importandroid.widget.EditText;
importandroid.widget.TextView;
importandroid.widget.Toast;
public class FileActivity extends Activity {private static final String
TAG = FileActivity.class.getName();
         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";
         writeToFile(textToSaveString);
                         String textFromFileString=
readFromFile();
                         if (
textToSaveString.equals(textFromFileString) )
         Toast.makeText(getApplicationContext(), "both string are
equal", Toast.LENGTH_SHORT).show();
                         else
         Toast.makeText(getApplicationContext(), "there is a
problem", Toast.LENGTH_SHORT).show();
                         Toast.makeText(this,"Text Saved
!",Toast.LENGTH_LONG).show();}
         private void writeToFile(String data) {
try {
OutputStreamWriteroutputStreamWriter = new
OutputStreamWriter(openFileOutput(FILENAME,
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 { InputStreaminputStream = openFileInput(FILENAME);
if ( inputStream != null ) {
         InputStreamReaderinputStreamReader = new
InputStreamReader(inputStream);
            BufferedReaderbufferedReader              =              new
BufferedReader(inputStreamReader);
            String receiveString = "";
            StringBuilderstringBuilder = new StringBuilder();
            while  (  (receiveString  =  bufferedReader.readLine())  !=
null ) {
stringBuilder.append(receiveString);}inputStream.close();
            ret = stringBuilder.toString();
            TextViewtv = (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



```
THIS DATA WRITTEN TO A FILE:hellohello
```



Hello World, FileActivity!

hi i am tayyaba

text file data:hi i am tayyaba

IV. Analysis And Results

Essentially we are trying to provide the standardization (through POSIX) and portability of Android applications on multiple operating systems. 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. [3]

- 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.

A. Quantified Feasibility Analysis

At this time Android covers the 53% of the Smartphone market share as shown in figure in 1. [23]. 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 9 it would be double. The statistical result is given in the table.1. This model focuses on the Android mobile users and Android developer through reusability and standardization

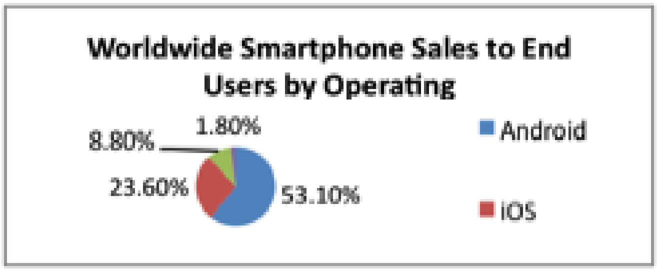
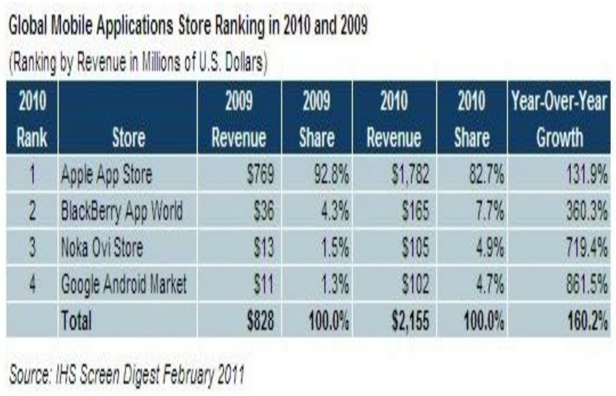


Fig.2. Worldwide Smartphone Sales to End User by Operating source: Gartner (February 2013)[23]

The Android applications market revenue increasing very rapidly [24]. 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	Nokia 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

Fig.3. Android market share [24]

The market of the desktop OS is very large as shown in the below table.1.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.8.

TABLE VII. MARKET SHARE ANALYSIS FOR ANDROID DEVELOPERS [25]

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 VIII. COMPARATIVE ANALYSIS OF APPLICATIONS DEVELOPMENT TIME AND MANPOWER FOR ANDROID DEVELOPERS

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

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.

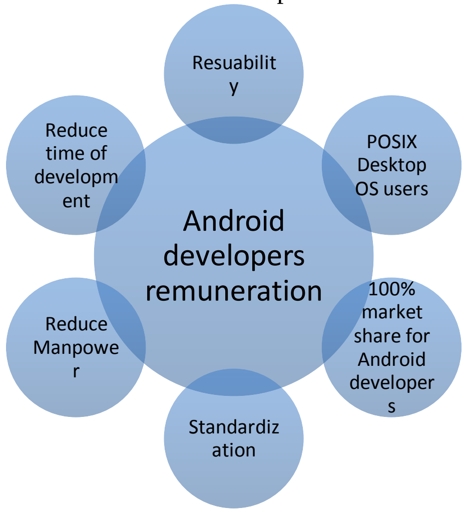

Resuability
POSIX Desktop OS users
100% market share for Android developers
Standardization
Reduce Manpower
Reduce time of development
Android developers remuneration

Fig.4. POSIX.1 thin layer Model resulting impact factors

V. 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 language conversion from java to standard C.

VI. 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. The Hardware acceleration for Graphics subsystem is also in require for completing implementation.

Second option is related to making the Android OS POSIX compliant. This is not an easy task as there are a lot of limitations of Android hardware restriction, Android devices variety plus DVM [32] 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. [33] This is only the idea but achievement needs more attention.

In the below 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.

The Android Runtime consists of the Dalvik virtual machine and the Java core libraries. The Dalvik virtual machine is an interpreter for byte code that has been transformed from Java byte code to Dalvik byte code. [30]

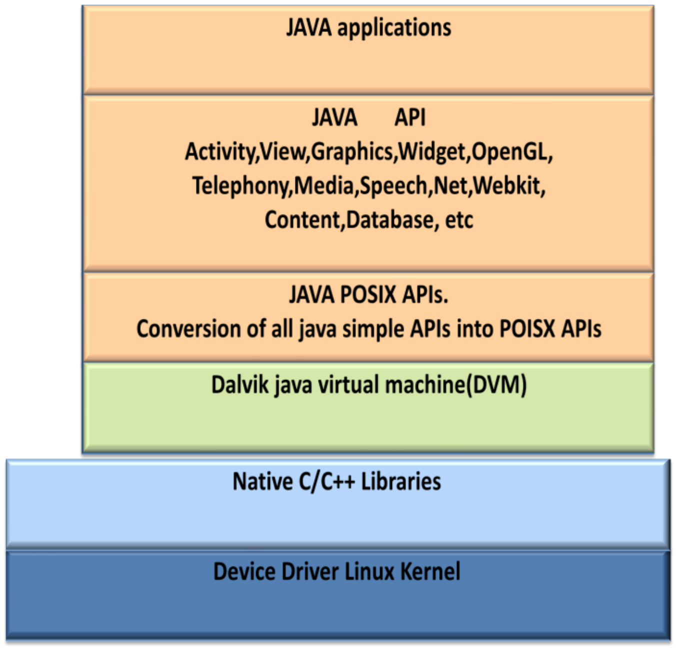

JAVA applications
JAVA API
Activity,View,Graphics,Widget,OpenGL,
Telephony,Media,Speech,Net,Webkit,
Content,Database, etc
JAVA POSIX APIs.
Conversion of all java simple APIs into POISX APIs
Dalvik java virtual machine(DVM)
Native C/C++ Libraries
Device Driver Linux Kernel

Fig.5. proposed Android OS POSIX compliant model

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 7 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 [31] layer which convert the alljava 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

VII. Conclusion

Currently, Android is the most popular operating system out of the several Linux based mobile operating systems (e.g.,Maemo) [4].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 and adds real-time capability and higher reusability

Acknowledgment

This research work is done under the supervision of Prof. Dr. Shoab Ahmad Khan. The Integrated development environments are Eclipse INDOGO for Android Application with Android virtual device (AVD) Intel Atom x86 with API level 10 and Xcode with gcc complier version is mention in previous Fig.1 at Macintosh OS.

References

[1] 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.

[2] Bill O. Gallmeister,POSIX. 4: Programming for the Real World, 1995.ppt.4, 19-20,22,23

[3] Donald A. Lewine, POSIX Programmer's Guide Writing Portable UNIX Programs with the POSIX.1 Standard, 1991,pp.16-17, 25

[4] E. Oliver, A Survey of Platforms for Mobile Networks Research. Mobile Computing and Communications Review, December 2008, pp. 56-63.

[5] Hassan Reza, and Narayana Mazumder, A Secure Software Architecture for Mobile Computing (2012 IEEE)

[6] Android. http://code.google.com/android/

[7] Canalys Report. http://www.canalys.org

[8] http://www.android.com/about/

[9] http://developer.android.com/about/versions/index.html

[10] POSIX. 1: ISO/IEC 9945-1:1990 IEEE Std. 1003.1-1990

[11] Donald A. Lewine, POSIX Programmer's Guide Writing Portable UNIX Programs with the POSIX.1 Standard, 1991,pp.31-36

[12] http://developer.android.com/sdk/eclipse-adt.html[Aug.20, 2011].

[13] http://www.ibm.com/developerworks/opensource/library/os-android-devel/[Aug.10, 2011].

[14] http://en.wikipedia.org/wiki/Android_ (operating_system) [Aug.10, 2011]

[15] http://en.wikipedia.org/wiki/Android_Market [Aug.20.2011].

[16] http://developer.android.com/guide/basics/what-is-android.html[Aug.20, 2011].

[17] Barra, Hugo (10 May 2011). "Android: momentum, mobile and more at Google I/O".The Official Google Blog. Retrieved 10 May 2011.

[18] http://www.google.com/support/androidmarket/developer/bin/answer.py ?answer=113475[Aug.6, 2011]

[19] http://www.android.com/about/[Aug, 6, 2011]

[20] Lawson, Stephen (17 March 2009). "Android Market Needs More Filters, T-Mobile Says". PC World.

[21] http://www.gartner.com/newsroom/id/2335616

[22] 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

[23] http://www.netmarketshare.com

[24] IEEE Std 1003.1b-1993 (Formerly known as IEEE P1003.4) (Includes IEEE Std 1003.1-1990)

[25] Kolin Paul, Tapas Kumar Kundu "Android on Mobile Devices: An Energy Perspective," 10th IEEE International Conference on Computer and Information Technology, 2010.

[26] Kyosuke Nagata,Saneyasu Yamaguchi "An Android Application Launch Analyzing System"

[27] IEEE Portable Applications Standards Committee, P1003.13: Infonnaiion Technology - Siandardized Applications Environment Profile - POSIX Real-time Application Support (AEP) (Draft 5) (Feb 1992).

[28] Namseung Lee, Sung-Soo Lim, "A Whole Layer Performance Analysis Method for Android Platforms", (2011 IEEE).

[29] Java POSIX APIs,

[30] http://bmsi.com/java/posix/posix-1.2.2/doc/index.html.Accessed March 2012

[31] 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.

[32] E. Cooper and R. Draves, "C threads". TR CMU-CS-88- 154, Carnegie Melloii University, Dept. of CS (1988).