Scientific Journal Impact Factor (SJIF): 1.711
International Journal of Modern Trends in Engineering and Research
ISSN (ONLINE) : 2349 - 9745 ISSN (PRINT) : 2393 - 8161

Package Analyzer
Automated Solution for Energy Inefficiency in Smartphone Applications

Raksha Shenoy S[1], Ranjitha D[2], Sai Pranathi Polisetti[3] Guide: Ms. Annapurna V K[4]
[1,2,3,4] CS & E, The National Institute Of Engineering, Mysore, Karnataka

Abstract - Smartphone applications' energy efficiency is vital, but many Android applications suffer from serious energy inefficiency problems. Locating these problems is labour-intensive and automated diagnosis is highly desirable. This work aims to address this challenge. Two common causes of energy problems were observed: missing deactivation of sensors or wake locks, and cost-ineffective use of sensory data [1]. With these findings an automated approach to solve energy problems in android applications is proposed. This method monitors sensor and wake lock operations to detect missing deactivation of sensors and wake locks by accessing the android's operating system packages. In this work only the ideal threads are terminated instead of an entire application. This approach is built as an android application.

Keywords - Wake locks, Sensory data, threads, JPF, Dynamic Tainting

I. INTRODUCTION

Android is an operating system developed by Google. There are many bugs in this operating system. One of the problems in android is the occurrence of deadlock. One more common problem is hanging of the phone. Since context switch between processes does not take place in orderly manner there might be situation in which the phone hangs during the usage of multiple applications at once. Android phones come with a host of hardware components embedded in them, such as Camera, Media Player and Sensor. Most of these components are exclusive resources or resources consuming more memory/energy than general. And they should be explicitly released by developers. Missing release operations of these resources might cause serious problems such as performance degradation or system crash. These kinds of defects are called resource leaks.

II. SYSTEM MODEL

The system is implemented using these modules.
2.1 Application Execution and State Exploration
2.2 Missing Sensor or Wake Lock Deactivation
2.3 Sensory Data Utilization Analysis

2.1 Application Execution and State Exploration

Android applications are mostly designed to interact with Smartphone users. Their executions are often triggered by user interaction events. Typically, an Android application starts with its main activity, and ends after all its components are destroyed. During its execution, the application keeps handling received user interaction events and sys-tem events (e.g., broadcasted events) by calling their handlers according to Android specifications. Each call to an event handler may change the application's state by modifying its components' local or global program data. As such, in order to execute an application and explore its state space in JPF, the following needs to be done: (1) generate user interaction events, and (2) guide JPF to schedule corresponding event handlers.

2.2 Missing Sensor or Wake Lock Deactivation

This section shows how to detect energy problems when exploring an application's state space. As mentioned earlier, missing sensor or wake lock deactivation is one common cause of energy problems. This shares some similarity with traditional resource leak problems, where a program fails to release its acquired system resources (e.g., memory blocks, file handles, etc.) Resource leak problems can cause system performance degradation (e.g., slower response), and similarly missing deactivation of sensors or wake locks can also waste valuable battery energy. Besides, according to Android process management policy, sensors and wake locks are not automatically deactivated even when the application components that activated them are destroyed (e.g., onDestroy() handler is called). Based on the preceding state exploration efforts, existing resource leak detection techniques are adopted to detect missing sensor or wake lock deactivation.

2.3 Sensory Data Utilization Analysis

During an Android application's execution, its collected sensory data are transformed into different forms and consumed by different application components. There is a need to track these data usages for energy efficiency analysis. This will be done at the byte code instruction level by dynamic tainting.

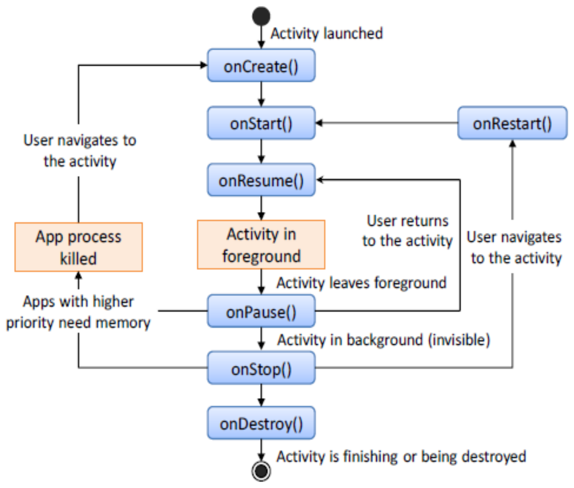
Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity in foreground
User returns to the activity
User navigates to the activity
Activity leaves foreground
Apps with higher priority need memory
onPause()
Activity in background (invisible)
onStop()
onDestroy()
Activity is finishing or being destroyed

Figure 1. An activity's lifecycle diagram

Every screen in an android application is called an activity. Each activity can exist in any of the following sates, thus called the lifecycle of an activity. The lifecycle of an activity is as follows as shown in fig(1). It starts with a call to onCreate() handler, and ends with a call to onDestroy() handler. An activity's fore-ground lifetime starts after a call to onResume() handler, and lasts until onPause() handler is called, when another activity comes to foreground. An activity can interact with its users only when it is at foreground. When it goes to background and becomes invisible, its onStop() handler would be called. When the users navigate back to a paused or stopped activity, that activity's onResume() or onRestart() handler would be called, and the activity would come to foreground again. In exceptional cases, a paused or stopped activity may be killed for releasing memory to other applications with higher priorities.

III. PREVIOUS WORK

Android applications are not energy efficient for two reasons. The reasons are that the Android framework exposes hardware operation APIs to developers. Hence the amateur developers will misuse the resources. Secondly, Android applications are mostly developed by small teams without dedicated quality assurance efforts. This will lead drop in the battery life.
The drawback of this system is that the sensing operations are usually energy consumptive, and limited battery capacity always restricts such an application's usage. The hardware misuse could easily lead to unexpectedly large energy waste and locating energy problems in Android applications is difficult. In the existing android applications that have been developed to save battery, the installed application will kill the running applications to conserve battery. This will force the user to reboot the system to run the applications again when needed.

IV. PROPOSED METHODOLOGY

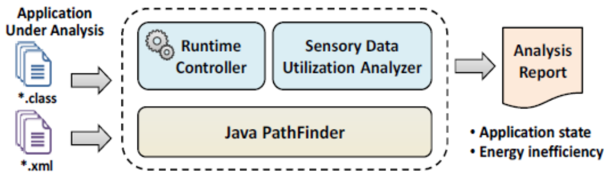

Fig 2.System overview

When an android application is coded, the following files come into existence namely, *.class file,*.xml file and an android manifest file. The *.class file holds the java byte code which contains the logic of the application, the *.xml file is the configuration file which holds information about the user interface and lastly the manifest file which holds all the permissions taken by the application from the OS.
After research two main causes for energy inefficiency are identified, namely: missing deactivation of sensors and wake locks and cost ineffective use of sensory data. In order to analyze the sensory data and missing wake locks we access the packages of all the installed applications as well as the packages of operating system. By using the analysis report generated by JPF(Java Path Finder) the ideal threads of the applications are identified. Once identified these threads are checked for existence of parent process and if so are terminated without terminating the entire application thereby preventing unwanted energy consumption.

V. SIMULATION/EXPERIMENTAL RESULTS

This approach is simulated as an android application with five activities where these activities give the list of packages installed and the ones of the operating system. For each package we identify the list of services, activities, receivers and permissions. These processes are classifies as daemon or non-deamon so that don't modify any daemon processes as it will lead to corruption of the operating system. The application analyzes the analysis report generated by JPF as set by the timer defined by the user to identify ideal threads. Once they are identified and listed the developed application will check if the thread has a parent process, if not the thread is terminated.
The result is that the application successful terminates all the ideal threads that drain the battery.

VI. CONCLUSION

Battery efficiency is a very important factor in an android system. There are two types of coding phenomena that commonly cause energy waste: missing sensor or wake lock deactivation, and sensory data underutilization. Based on these findings, an approach for automated energy problem diagnosis in Android applications is identified through an analysis report generated by JPF. This method identifies idle threads that consume battery and terminates it with an exception of daemon process threads.

VII. FUTURE SCOPE

In future the application can be protected from malicious security attacks and stringent security measures can be employed to protect from malicious attacks. The application can be hosted on the server so that the developer can monitor the user and the usage. The application can be modified to generate bug reports so that it can be improvised accordingly. The application can be scaled to be compatible with higher API's that come in the market.

REFERENCES

[1] Authors:Yepang Liu, Chang Xu, S.C. Cheung and JianLü "GreenDroid: Automated Diagnosis of Energy Inefficiency for Smartphone Applications.",IEEE, 2014.