计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(1):256−261 [doi: 10.15888/j.cnki.csa.006715]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

# Android 唤醒锁检测及功耗优化机制[①]

来庆波, 陈　博, 茆　蕾, 汪福翔, 司俊超

(中国科学技术大学 软件学院 (苏州市中国科学技术大学苏州研究院), 苏州 215123)

**摘　要**: Android 操作系统提供了唤醒锁机制用于避免系统进入休眠状态. 但若存在唤醒锁的误用, 将导致设备能耗的加剧并严重影响用户体验. 本文分析总结了常见的唤醒锁误用类型及原因, 并实现了一款安卓应用, 用以检测当前系统中持有唤醒锁的进程, 并根据某些策略释放误用的唤醒锁. 结合具体实机测试数据, 本文提出的检测机制及功耗优化方法, 可有效降低功耗, 提高用户体验.

**关键词**: 安卓休眠机制; 唤醒锁误用; 唤醒锁检测; 功耗优化

引用格式: 来庆波,陈博,茆蕾,汪福翔,司俊超.Android 唤醒锁检测及功耗优化机制.计算机系统应用,2019,28(1):256–261. http://www.c-s-a.org.cn/1003-3254/6715.html

## Detection and Optimization of WakeLock on Android Platform

LAI Qing-Bo, CHEN Bo, MAO Lei, WANG Fu-Xiang, SI Jun-Chao

(School of Software Engineering, University of Science and Technology of China (Suzhou Institutes, University of Science and Technology of China), Suzhou 215123, China)

**Abstract**: The Android PowerManager provides a programming interface called WakeLock to protect critical computation from system hibernation. The WakeLock acquisition and release in the application can have a significant impact on power dissipation. However, the misuse of WakeLock increases the energy dissipation of the device and seriously affects the user experience. This article analyzes and summarizes the types and causes of WakeLock misuse. Furthermore, we developed an Android application that can detect and optimize the WakeLock misuses to reduce the power dissipation.

**Key words**: Android PowerManager; WakeLock misuse; WakeLock detection; power optimization

社交网络的快速发展, 使得智能手机已深度融入生活. 另一方面, 频繁的交互操作使得手机功耗问题凸显. 为延长待机时间, 许多智能手机系统使用睡眠策略来节省电量, 但有些应用需要手机在某些关键计算时保持运行状态. 例如银行应用程序, 当用户在线转账时, 交易可能需要一段时间才能完成. 若手机在等待服务器消息时被置于休眠, 造成没能及时响应, 则转账将会失败. 为解决此问题, Android 系统设计了唤醒锁, 使某些硬件在计算时保持运行状态 .

然而, 在现实中, 很多开发者存在滥用唤醒锁的问题, 在不必要的时候仍占用唤醒锁, 这加重电能消耗, 严重影响用户体验.

为解决唤醒锁误用带来的电耗增加问题, 本论文研究了常见的唤醒锁误用类型, 在此基础上, 总结出了两种判定误用的策略: 第一种是根据 CPU 占用率模式, 第二种是根据源码分析建立黑名单. 并利用 PowerManager 提供的服务接口来检测与释放唤醒锁.

## 1　背景知识介绍

### 1.1　Linux 休眠机制简介

Linux 存在两种电源管理方案: 高级电源管理 (APM) 和高级配置电源界面 (ACPI). 缺省情况下运行

① 基金项目: 苏州市科技计划项目 (SYG201731)
Foundation item: Science and Technology Plan of Suzhou Municipality (SYG201731)
收稿时间: 2018-06-03; 修改时间: 2018-06-27; 采用时间: 2018-07-25; csa 在线出版时间: 2018-12-26



© 中国科学院软件研究所　http://www.c-s-a.org.cn

ACPI. ACPI 在节电方面有很多机制, 可以让你把机器处于 Suspend(悬挂) 或 Standby(备用) 状态. 还可以让你把外设 (如: 显示器、显卡、PCI 总线) 单独断电.

Linux 的休眠机制可概括为以下三个步骤:

(1) 冻结用户态进程和内核态任务;

(2) 调用注册的设备的 Suspend 的回调函数, 其调用顺序是按照驱动加载时的注册顺序;

(3) 休眠核心设备和使 CPU 进入休眠态. 冻结进程是内核把进程列表中所有的进程的状态都设置为停止, 并且保存下所有进程的上下文.

### 1.2 Android 休眠机制简介

标准的 Linux 电源管理机制是给带有外接电源的电脑设计的, 睡眠机制有一些缺陷 (如, 所有模块必须同时睡眠或唤醒), 这会导致不必要的能耗. 这些机制并不适用于电池容量有限的移动平台, 因此 Android 在 Linux 休眠机制的基础上衍生出了独特的 WakeLock 机制来管理和节省电源.

其基本原理如下: 当启动一个应用程序的时候, 它可以申请一个 wake_lock, 每当申请成功之后都会在内核中注册一下 (通知系统内核, 现在已经有锁被申请), 当应用程序在某种情况下释放 wake_lock 的时候, 会注销之前所申请的 wake_lock. 特别要注意的是: 只要系统中有一个 wake_lock, 系统此时都不能进行睡眠. 只有当系统中所有的 wake_lock 都被释放之后, 系统才会进入真正的睡眠状态.

图 1 描述了 Android 唤醒锁调用的内部设计. 内核唤醒锁是只能通过 Linux 内核内部获取或释放. Android 开发人员无法直接控制内核唤醒锁, 但应用程序可能会间接触发这些唤醒锁. 当通过 PowerManager 的 API 创建和获取唤醒锁时, 该请求将通过绑定 IPC 机制传递到名为 PowerManagerService 的系统服务. 如果该请求是 PARTIAL_WAKE_LOCK 实例, PowerManagerService 将调用 Android_os_power.cpp 的方法, 通过 JNI 读取/写入 Linux 内核系统处理请求. 否则, PowerManagerService 本身将以不同于部分唤醒锁定的方式处理请求. PowerManagerService 记录 PARTIAL_WAKE_LOCK 的数目. 当应用程序获取 PARTIAL_WAKE_LOC 实例时, PowerManagerService 会将计数增加 1, 并在释放唤醒锁时减少 1. 最后, 如果 PARTIAL_WAKE_LOCK 的计数为零, PowerManagerService 将通知 Linux 电源管理系统, 设备已经可以进入休眠模式.

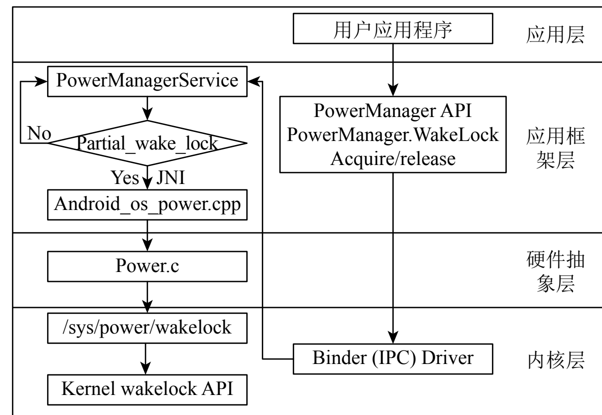


图 1 Android 唤醒锁调用的内部设计图

## 2 相关工作

近年来很多人开展了有关唤醒锁的研究. 对于误用唤醒锁检测方面, Pathak 等人进行了第一次的相关研究, 并采用数据流分析技术检测唤醒锁泄漏[1]. 后来相继出现了其他静态和动态分析技术, 为唤醒锁误用的检测提供了众多方法. 例如: Vekris 等人提出了一种验证 Android 应用程序中是否存在唤醒锁泄漏的静态分析技术[2]; 以及在一篇介绍 WLCleaner 的论文中, 设计了一种动态分析技术, 检测由应用程序引起的唤醒锁误用, 并采取措施纠正这些误用[3]. 这些研究集中于研究唤醒锁引起的能耗问题. 在资源管理和泄漏检测方面, Jindal 等人通过研究识别了 Android 设备驱动程序中的四种睡眠冲突类型, 并提出了一种避免这种冲突的运行时调试系统[4]. Relda 采用资源安全策略检查的思想来检测 Android 应用程序中的资源泄漏, 包括唤醒锁资源的泄漏[5].

据 Android 官方称 Android 8.0 (API26) 为了改善电池寿命, 引入了一种新的机制[6]. 当某个应用程序进入缓存状态、没有活动的组件时, 系统会自动释放此应用程序持有的所有唤醒锁. 但是根据 Google 发布的数据, Android 8.0 目前的市场占有率仅有 5% 左右; 且目前很多在用的手机, 制造商不会提供 Android 8.0 的系统. 所以研究 Android 8.0 之前版本的功耗优化目前仍具有现实意义.

## 3 唤醒锁误用原因分析

唤醒锁误用是指程序本来应该释放唤醒锁而因为各种原因未能正确释放唤醒锁. 导致这一问题的原因



© 中国科学院软件研究所 http://www.c-s-a.org.cn

非常复杂, 下面分析几种常见的情景.

### 3.1 某些运行路径上未能释放唤醒锁

由于 Android 程序属于事件驱动型的, 不同的用户交互行为或不同的硬件环境会导致不同的运行路径, 在代码动态运行过程中, 会出现开发者未能预料到的运行路径. 即使经验丰富的开发者也难以避免这一情况. 原因在于: 触发这一情况的条件比较苛刻. 只有在特定的使用场景下才会发生. 如, 特定的用户交互行为或特定的使用环境 (如 GPS 信号弱), 而这些情况是很少出现的.

如下面的代码段:

```
1. try{
2.   wkl.acquire();
3.    run_cal();
4.   wkl.release();
5. }catch(Exception e){
6.    system.out.println(e);
7. }finally{
8.   //the end
9. }
```

上面的示例代码是一段典型的可能存在唤醒锁泄漏的情况, 关键任务 run_cal() 被唤醒锁保护, 以保证 CPU 在执行此任务时不会进入休眠状态. 按照开发者的设想, 执行完 run_cal() 任务后, 唤醒锁就被会被释放掉. 但是如果在 run_cal() 执行过程中, 发生了异常错误, 比如数学运算时产生了除零错误、传送文件时网络断开、打开的文件不存在、GPS 信号弱等, 这时抛出的异常被 catch 语句块捕获并产生相应的处理. 导致 wkl.release 语句不能被执行, 从而使 CPU 不能进入休眠状态. 一个解决方法是: 在 finally 语句块中加入 wkl.release() 语句. 这样, 即使 run_cal() 任务在执行过程中发生了异常, 唤醒锁也会被正确释放.

在 youku 3.0[7]中, 开发者仅在 DownloadListenerImpl 类的 onFinished() 回调函数中释放了唤醒锁, 这样当在下载过程中出错或取消下载时, 就会发生唤醒锁泄漏. 正确的做法是在 onCancel() 和 onException() 两个回调函数中增加释放唤醒锁的语句.

在 mytracks[8]中, 程序在后台中执行记录踪迹任务时, 需要获取唤醒锁. 当任务完成正常返回时, 调用 onPostExecute() 函数释放唤醒锁. 当如果在任务执行时, 发生了用户取消任务或程序异常, 就会发生唤醒锁泄漏. 正确的做法是在 onCance() 回调函数和 finally 语句块中增加释放唤醒锁的语句.

### 3.2 仅在用户回调函数中释放唤醒锁

Android 程序的入口不是单一的 main 函数, 而是若干个回调函数. 这些回调函数可分为系统回调函数和用户回调函数. 系统回调函数包含组件的生命周期函数 (如: onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy()) 和一些安卓框架函数 (如 Thread 类里的 run()). 用户回调函数通常是 UI 事件触发的, 如鼠标点击事件 onClick()、触摸事件 onTouch()、键盘事件 onKey()、状态变换事件 onFocusChange() 等.

如果开发者仅仅在用户回调函数中设置了释放唤醒锁的语句, 而未在系统回调函数中设置释放语句. 释放唤醒锁的语句很可能因为用户没有执行预设动作而未能触发. 这一情况在经验不足的开发者中经常出现. 如, 在 BaiduMap 5.0 中开发者仅在用户回调函数中设置了释放 LocationManager 和 AudioManager 的语句.

在相应的系统回调函数 (如 onPause(), onStop()) 中增加释放唤醒锁的语句, 可有效解决此类问题.

### 3.3 在不合适的生命周期函数上释放唤醒锁

安卓的四大组件都有相应的一组函数来处理生命周期事件. 图 2 描述了 Activity 组件的生命周期[9].

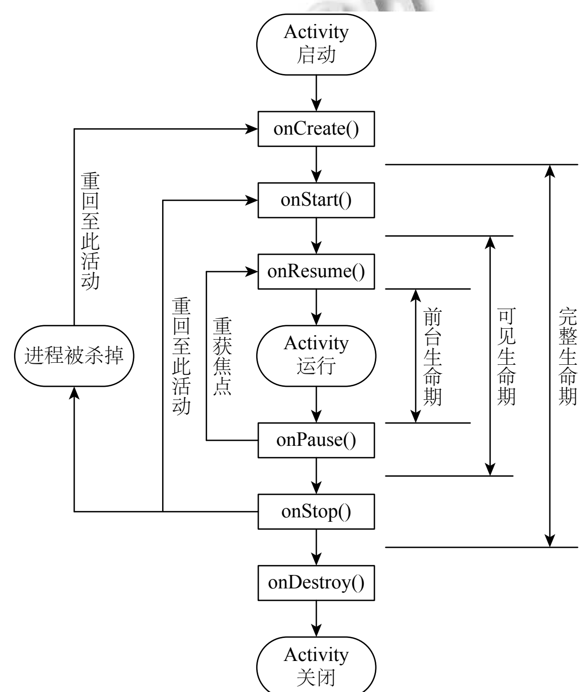


图 2 Activity 生命周期示意图



© 中国科学院软件研究所 http://www.c-s-a.org.cn

Activity 生命周期有三个嵌套循环, 分别对应完整生命期、可见生命期、前台生命期.

生命周期函数都有固定的调用顺序. 如在 Activity 组件被创建时, onCreate()、onStart()、onResume() 会相继被调用. 如果开发者对上述生命周期的调用顺序理解不足, 可能会造成以下误用: 唤醒锁的释放早于唤醒锁的获取, 这将会释放不存在的唤醒锁, 导致系统崩溃. 如 ConnetBot[10]就存在此类误用.

另外, 安卓系统在前台退出一个进程时, 为方便下次启动更快速, Android 后台并没有完全退出, 在内存足够的情况下系统还会把它留在内存里. 只有当手机内存不足以启动一个新进程时, Android 才会把不用的进程彻底停掉. 也就是说, 在前台退出进程时 onDestroy() 并没有被调用. 如果开发者对上述知识理解不足, 可能会仅在 onDestroy() 中释放唤醒锁而没有在 onPasuse() 中释放唤醒锁. 这会导致系统长时间不能进入休眠状态, 从而引起电耗增加.

### 3.4 唤醒锁的类型设定不合理

Android 系统提供了四种唤醒锁. PARTIAL_WAKE_LOCK: 保持 CPU 运转, 屏幕和键盘灯有可能是关闭的. SCREEN_DIM_WAKE_LOCK: 保持 CPU 运转, 允许保持屏幕低亮度显示, 允许关闭键盘灯. SCREEN_BRIGHT_WAKE_LOCK: 保持 CPU 运转, 允许保持屏幕高亮显示, 允许关闭键盘灯. FULL_WAKE_LOCK: 保持 CPU 运转, 保持屏幕高亮显示, 键盘灯也保持常亮. 使用 PARTIAL_WAKE_LOCK 锁, 无论屏幕的状态是什么, 或者用户按了电源按钮, CPU 都会继续工作. 如果是其它的唤醒锁, 设备会在用户按下电源钮后停止工作进入休眠状态.

表 1 Android 唤醒锁类型

| 标记值 | CPU | 屏幕 | 键盘 |
|---|---|---|---|
| PARTIAL_WAKE_LOCK | 开启 | 关闭 | 关闭 |
| SCREEN_DIM_WAKE_LOCK | 开启 | 变暗 | 关闭 |
| SCREEN_BRIGHT_WAKE_LOCK | 开启 | 变亮 | 关闭 |
| FULL_WAKE_LOCK | 开启 | 变亮 | 变亮 |

如果开发者对以上所述的唤醒锁类型理解不到位, 使用了不合适的唤醒锁类型, 就会造成误用. 例如在记步软件中, 进程在后台获取位置信息的时候并不需要屏幕保持常亮, 使用 PARTIAL_WAKE_LOCK 保持 CPU 运转就可以了. 如果开发者使用了 FULL_WAKE_LOCK 类型的锁让屏幕保持常亮, 就会造成不必要的电量消耗.

## 4 唤醒锁检测及优化的实现

### 4.1 Wlresolver 概述

本文研究了唤醒锁检测及电耗优化机制, 并开发一款 Android 应用程序“Wlresolver”, 来实现这些机制.

在 Android 系统中可以通过 PowerManager 提供的服务接口来获取与释放 WakeLock, 然而用户无法直接访问 WakeLock 的相关数据. PowerManagerService 位于应用框架层, 提供与电源管理相关的一系列接口, 是整个系统的电源管理核心, 在应用框架层之下的硬件抽象层有一个 power.c 文件, 通过上层传递的参数, 向/sys/power/wake_lock 或者/sys/power/wake_unlock 文件写入数据来与内核进行通信, Wlresolver 在 Android 系统层次结构中的位置如图 3.

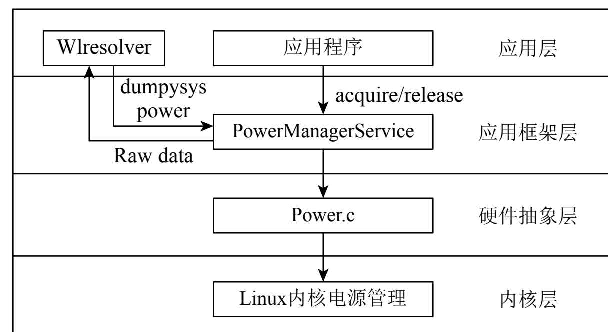


图 3 Wlresolver 在 Android 系统层次结构图中的位置图

### 4.2 主要模块设计及唤醒锁检测机制

Wlresolver 主要由三个部分组成: WakeLock 检测模块、WakeLock 处理模块、用户交互界面模块.

当手机屏幕关闭后延时触发处理模块从检测模块中获取数据并处理其中误用的 WakeLock. 当屏幕关闭动作产生时, Android 系统会向所有应用发出广播, 在软件中添加广播接收器接收系统广播, 当收到系统发出的屏幕关闭广播时触发启动后台服务的代码, 此时处理模块开始运行, 当屏幕点亮时, 另一个广播接收器接收到广播, 关闭处理模块.

WakeLock 检测模块通过 adb shell 命令“dumpsys power”获取 WakeLock 相关的信息, 如图 5, 并且将得到的原始文本数据转换所成需要的数据, 转换出的数据包含三个部分的信息: WakeLock 对应的软件名称、WakeLock 类型、uid、pid.

### 4.3 工作流程及唤醒锁释放策略

Wlresolver 的工作流程图如图 6 所示.



© 中国科学院软件研究所 http://www.c-s-a.org.cn

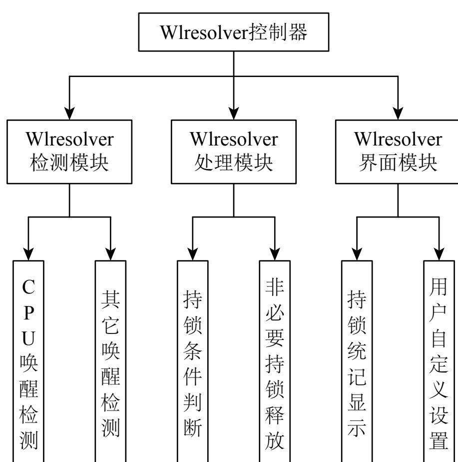


图 4 Wlresolver 的主要模块示意图

```
dumpsys power| grep -i wake
  mWakefulness=Awake
  mWakefulnessChanging=false
  mWakeLockSummary=0x1
  mLastWakeTime=63317566 (326290 ms ago)
  mHoldingWakeLockSuspendBlocker=true
  mWakeUpWhenPluggedOrUnpluggedConfig=true
  mWakeUpWhenPluggedOrUnpluggedInTheaterModeConfig=false
  mDoubleTapWakeEnabled=false
Wake Locks: size=1
  PARTIAL_WAKE_LOCK                'UMSE PowerTest' (uid=100
62, pid=27347, ws=null)
```

图 5 Wlresolver 获取数据

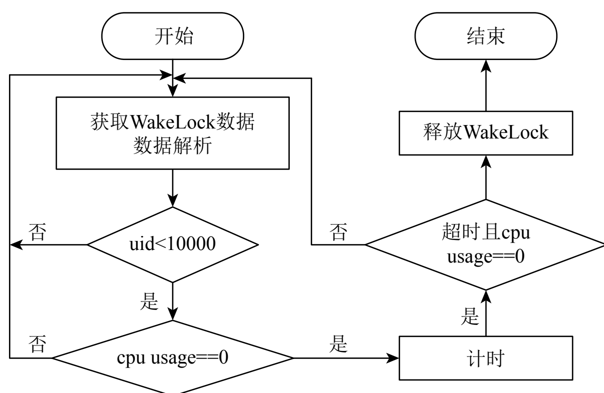


图 6 Wlresolver 工作流程图

当程序被触发时, 首先导出系统中的 WakeLock 数据, 并将原始数据解析为我们所需要的数据. 接着过滤掉 uid>10000 的数据, 因为 uid>10000 的是系统程序, 系统程序在屏幕关闭时可能需要一直运行, 强行关闭系统程序可能导致系统崩溃.

为了判断应用程序是否有误用, 第一个策略是: 检测持有唤醒锁的进程的 CPU 使用率. 如果某进程一直持有唤醒锁, 但是在某段时间内 (如 50 s) 未使用 CPU, 可以判定为该进程申请的唤醒锁为非必要的持有, 可以将其释放. 因为根据统计数据[11], 有 70% 的持有唤醒锁的进程会一直使用 CPU, 其它的持有唤醒锁的进程也会在 5 s 以内使用 CPU(由于程序运行中可能会因为一些操作中断例如 I/O 或者用户交互, 在这种情况下它将在短时间内恢复运行).

另一个释放唤醒锁的策略是: 先利用反向工程取得应用程序的源码, 然后利用 elite 进行静态分析[12], 预先识别出存在误用的应用程序. 进而在 Wlresolver 中建立一个黑名单列表. 黑名单内的进程如果持续持有唤醒锁超过某个时间就释放该进程的唤醒锁.

## 5 评测

### 5.1 实验方法概述

为评估 Wlresolver 的使用效果和本文提出的优化策略的有效性, 我们开展了相关实验.

本文使用 HUAWEI GRA-TL00 型号手机作为测试平台, Android 版本为 5.0.1. 为了保证整个测试过程的一致性, 手机关闭了 wifi 信号、蜂窝信号等使用频次不确定的服务.

本文使用 Google 开发的 battery-historian 来检测电量消耗. Android 为了方便开发人员分析整个系统平台和某个进程在运行时间内的所有信息, 专门开发了 bugreport 工具. 在终端执行: adb bugreport > bugreport.txt, 即可生成 bugreport 文件. Google 针对 Android 5.0(api 21) 以上的系统开发了一个叫做 battery historian 的分析工具, 用来解析 bugreport.txt 文本文件, 并用 Web 图形的形式展现出来, 从而获得详细的电池耗电情况[13].

本文从华为应用商店下载了 70 个安卓应用程序, 并且从 Google Code、Github、SourceForge 等开源仓库下载了 40 个开源软件, 进行测试.

### 5.2 实验结果与分析

在手机中运行测试软件, Wlresolver 检测到了 6 个进程 (pedometer、悦动圈、网易云音乐等) 在运行时持有唤醒锁. Wlresolver 开启服务后, 有一个进程 (悦动圈) 被杀掉. 说明根据 Wlresolver 的策略判定悦动圈持有的唤醒锁为误用. 我们对悦动圈的使用情况进行了分析, 发现此进程在用户不记步的情况下仍然持有唤醒锁, 消耗大量电能, 确实存在误用的情况.

此后, 我们进行了电量消耗速率进行了多组测试. 选取了四组实验数据绘图, 四组实验在 Wlresolver 未开启服务的情况下分别运行 1 h/5 h/12 h/24 h, 接着在



© 中国科学院软件研究所 http://www.c-s-a.org.cn

Wlresolver 开启服务的情况下分别运行 1 h/5 h/12 h/24 h. 使用 battery-historian 获得电量消耗数据, 进行电量消耗对比. 从图 9 可以看出开启服务后电量消耗速率均有所下降, 平均下降了 1.85%. 下降幅度取决于存在唤醒锁误用的进程数量. 如果手机运行时存在唤醒锁误用的进程越多, 电耗下降效果就会越明显.

图 7 Wlsolver 运行时截图

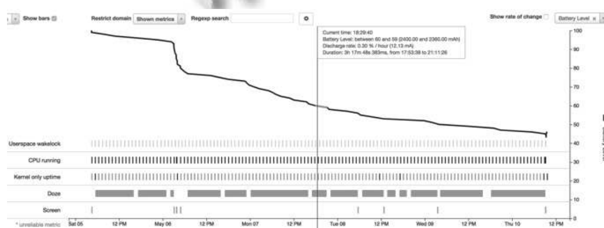

图 8 利用 battery-historian 显示电池消耗数据

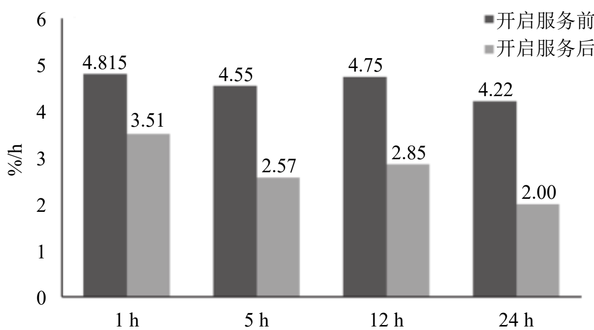


图 9 根据 battery-historian 数据绘制的电耗速率对比图

## 6 结语

本文首先介绍了 Linux 和 Android 的休眠机制. 然后分析了 Android 开发者在实际编码过程中可能存在的唤醒锁误用类型及原因. 最后研究了唤醒锁检测及电耗优化机制并通过 Wlresolver 实现了上述机制, 开启服务后, 软件会自动按照既定策略运行, 不需要用户手动干预; 经过实验验证, Wlresolver 启动服务后成功的清除了误用的唤醒锁, 系统电量消耗有所下降.

## 参考文献


1 Pathak A, Jindal A, Hu YC, *et al*. What is keeping my phone awake?: Characterizing and detecting no-sleep energy bugs in smartphone apps. Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services. Low Wood Bay, Lake District, UK. 2012. 267–280.

2 Vekris P, Jhala R, Lerner S, *et al*. Towards verifying android apps for the absence of no-sleep energy bugs. Proceedings of the 2012 USENIX Conference on Power-Aware Computing and Systems. Hollywood, CA, USA. 2012. 3.

3 Wang XG, Li XF, Wen W. WLCleaner: Reducing energy waste caused by wakelock bugs at runtime. 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing. Dalian, China. 2014. 429–434.

4 Jindal A, Pathak A, Hu YC, *et al*. Hypnos: Understanding and treating sleep conflicts in smartphones. Proceedings of the 8th ACM European Conference on Computer Systems. Prague, Czech Republic. 2013. 253–266.

5 Torlak E, Chandra S. Effective interprocedural resource leak detection. 2010 ACM/IEEE 32nd International Conference on Software Engineering. Cape Town, South Africa. 2010. 535–544.

6 Android 8.0 behavior changes. https://developer.android.google.cn/about/versions/oreo/android-8.0-changes?hl=fr. [2018-07-31]

7 Youku. https://www.youku.com/.

8 Qin D. MyTracks. https://baike.so.com/doc/5096919-5325222.html. [2015-01-01]

9 McLemore M. Activity Lifecycle. https://docs.microsoft.com/zh-cn/xamarin/android/app-fundamentals/activity-lifecycle/. [2018-02-28]

10 ConnectBot.https://baike.so.com/doc/7657808-7931903.html.

11 Yoon C, Kim D, Jung W, *et al*. AppScope: Application energy metering framework for android smartphones using kernel activity monitoring. Proceedings of the 2012 USENIX Conference on Annual Technical Conference. Boston, MA, USA. 2012. 36.

12 Liu YP, Xu C, Cheung SC, *et al*. Understanding and detecting wake lock misuses for android applications. Proceedings of the 201624th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Seattle, WA, USA. 2016. 396–409.

13 battery-historian V2.0 的数据获取及参数分析. https://blog.csdn.net/liangxy2014/article/details/78311938. [2017-10-22]




© 中国科学院软件研究所 http://www.c-s-a.org.cn