Energy-Aware Task Killer

EE202B – 01/2017 to 04/2017

Pranshu Bansal and Saurabh Deo
Prof. Mani Srivastava

1. Abstract

As smart phone devices become more advanced, the operating system (OS) tries to become even more clever about scheduling and descheduling applications. However, this cleverness is sometimes at a cost to the user. For example, consider a phone running two applications. The problem arises when the OS deschedules an app to preserve battery, and continues running other apps in the background. This is often done without alerting the user. The user may have wanted preserve the app that was killed, and instead kill another app that appears to the OS as a more critical application. As such, we have built an app that allows users to have control over which apps get killed when their battery reaches certain, predefined thresholds. Future work could involve adding features to the app such as allowing users to select their own critical battery thresholds.

2. Introduction

Consider the case when a user performing athletic activity while running two applications: a health monitoring app (Google Fit), and generic music playing app (Spotify).

They are very interested in tracking how they are improving over the course of the year. As such, they would really like the Google Fit app to continue running in the background. They are less concerned with listening to music, as the gym that they are working out in plays a radio.

When the battery falls below a certain threshold (e.g. 10%), the health monitoring app may be descheduled as the OS deems it to be a less critical app. This conflicts with the users interests as they prefer to continue to track their fitness rather than listen to music.

According to our background research, no apps currently exist that enable this kind of control over scheduling.

In developing this project, we learned about the following areas and contributed as follows:

- Built and verified functionality of an app that can enable users to set priorities for various apps at various battery thresholds, such that the apps they care about are not killed by the OS.
  - Must be adoptable (safe to use, not exposing dangerous APIs)
  - Must not require developer knowledge to install (cannot require recompilation of kernel)
- Learned how to develop an app using Android Studio.
- Downloaded, compiled, and flashed the AOSP (Mako) onto a Nexus device.
- Experimented with converting our app to be a system application.
- Discovered the various issues associated with establishing correct privileges required to run shell scripts and kill apps.

3. Background

We considered four approaches to implement our idea: developing a custom app in Android studio, developing solutions using shell scripts, using existing frameworks and applications to build our system, and modifying the kernel to implement a custom scheduler or exposing API's to kill tasks. The idea we decided to implement was a custom app, with a custom shell script to support killing low-priority applications. We also explored implementation schema in the kernel space, and other frameworks provided by existing apps.

3.1 Custom App with Android Studio

We decided to create our own custom application as we could not find any existing frameworks or applications suitable for our needs. However, we require root access to run this application. This is to be expected of applications attempting to interfere with the way that other apps function.

3.2 Shell Scripts

Or system required us to build a shell script to facilitate killing a task. This is because of the Android convention. Android apps should never be able to be killed by another app, and therefore, no such API is exposed to developers. This posed an interesting challenge as we had to modify manifests and other Android permissions to transfer the shell script from a personal computer to the Android device.

3.3 Existing Applications

We mainly used Tasker [1], but we also did a brief review on other apps such as Condi, Automate, MacroDroid, and AutoMelt [2]. These apps fell into three broad categories.

3.3.1 Trigger Based Frameworks

This category included apps such as Tasker and Condi. The idea is that the user sets up triggers and control flows to decide which apps to kill, given certain conditions (e.g. state of charge (SoC) of the battery, time of day, geolocation, etc). However, these control flows are generally difficult for users to build if they lack a programming background.

Additionally, there were bugs in Tasker. For example, if a user no longer wants tasker to kill certain apps, the user cannot simply turn Tasker off. They must restart their smart phone. This is very inconvenient, and we therefore did not rely on the Tasker framework.

3.3.2 Flowchart Based Frameworks

Apps such as Automate allow users to build scheduling schema using visual flow diagrams. However, this also

suffers from "difficult to build" as explained in the Trigger Based Frameworks section above.

3.3.3 Event Based Scheduling

These apps focus on scheduling rather than descheduling of apps based on user defined events. As such, these apps are not suitable for our purposes.

3.4 Kernel Modifications

As stated in the introduction, we wanted to make sure that this app is useable by "non-developers". As such, we explored options in the Kernel space, but decided against making modifications to the scheduler. We cannot expect average users to rebuild their entire OS to run our app.

In particular, we experimented with Android Open Source Project (AOSP) and the Mako build (based on Android v4.1.1).

4. System Implementation

We built our app to run on a Nexus 4 device running stock Android v5.1.1 [3]. Our system implementation is nicely summarized in Figure 1. It shows that our app operates in three distinct phases: initialization, start, and run.

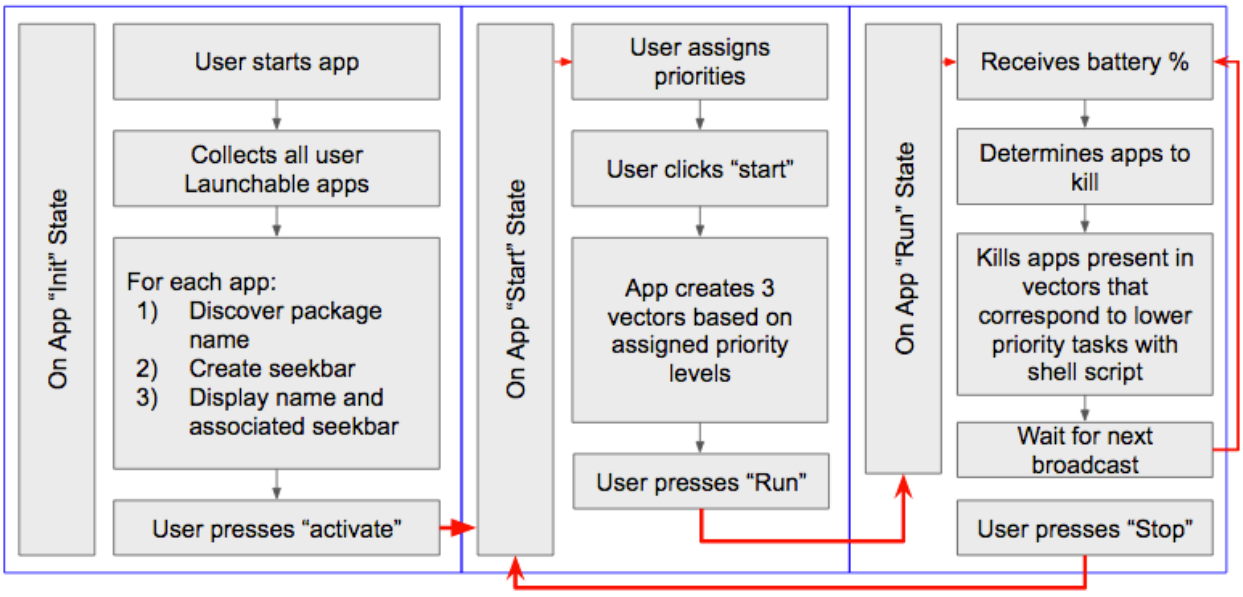

Figure 1: System diagram of how our app functions

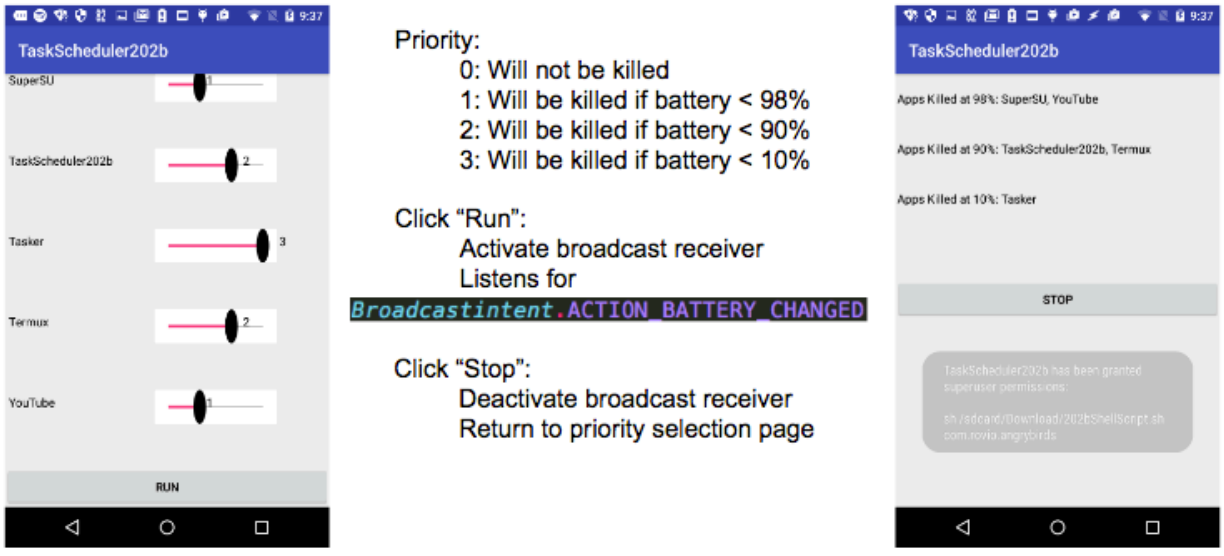

Figure 2: Front end of application.

4.1 Initialization Phase

This phase mostly deals with initializing the Start screen and obtaining a list of *launchable* apps. This list is populated via call to *getApplicationContext().getPackageManager()*and *PackageManager.GET_META_DATA*.

4.2 Start Phase

4.2.1 Custom SeekBar Class

Wanted an easy way for users to select priorities. However, these need to be discrete, and a typical seek bar only allows for continuous values. Additionally, seek bars do not display the current setting on the screen.

To get around these issues, we implemented a custom SeekBar (Figure 4) class called *dynamicSeekBar* (Figure 3) to enable these features.

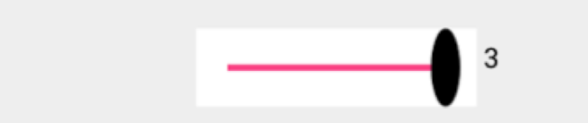

Figure 3: dynamicSeekBar. Note that it displays the value to the right and is a discrete value.

Figure 4: regular SeekBar

4.2.2 Run Button

Once the user is happy with their priority assignments, they can click the "Run" button pictured in Figure 1 to begin killing tasks deemed to be low priority, when the battery reaches certain critical threshold values.

The priorities are collected when the user clicks "Run". The entire screen is a table, so our app runs through the table (app by app) and populates three vectors: low priority, medium priority, and high priority. Low priority tasks are killed if the battery drops below 98%. Medium priority tasks are killed if the battery drops below 90%. High priority tasks are killed if the battery reaches below 10%. Anything not assigned a priority will not be killed by our app.

Once the vectors are passed to a broadcast receiver class, and the broadcast receiver is listening for the relevant broadcast, the app switches to the Run State.

4.3 Run State

A broadcast receiver listens for a broadcast transmitted by the Android OS. Once it receives a specific broadcast (in our case *Broadcastintent.ACTION_BATTERY_CHANGED*), the class calls a member function "onReceive()". After this function returns, the class is recycled.

As such, we had to make some modifications to the class to prevent the initially passed vectors from getting recycled. This was achieved by initializing the vectors to be static within the class.

```
static ArrayList<String> lowKillList;
static ArrayList<String> mediumKillList;
static ArrayList<String> highKillList;
static HashMap<String, String> appPackMap;
```

Figure 5: Code snippet from MyBR.java showing how we resolved the vectors getting recycled on each onReceive() call.

5. Challenges

We faced many challenges when building this app. A lot of these challenges were related to this being our first app building experience. We did manage to resolve most of the issues, and have learned a lot during the process.

5.1 Resolving Permission Denied Errors

The biggest issue to overcome was realizing that the code segment displayed in Figure 6 would not execute as expected. We expected the system to enter super user mode, and then run the shell script in super user mode.

```
process = Runtime.getRuntime().exec("su");
process = Runtime.getRuntime().exec("sh /sdcard/Download/202bShellScript.sh " + packagename);
```

Figure 6: Incorrect usage of shell command "su" to gain privileges required to run a shell script.

Instead, the application started a separate super user mode terminal session, and executed the next command in regular mode.

To resolve this, we used the code segment displayed in Figure 7.

```
exec("su -c sh /sdcard/Download/202bShellScript.sh " + packagename);
```

Figure 7: Correct usage of "su" command with "-c" flag to execute following arguments in super user mode.

5.2 System Application

When Tasker kills an app, we found that the app restarted itself in the background. One idea we had was to convert Tasker to a system app, and add functionality such that the OS would think the app we wanted to kill had restarted, when in fact it was dead.

This caused our entire system to fail as we had several apps that we did not want to kill being killed, apps disappearing from the phone and other such issues.

We resolved this by flashing a new version of stock Android (v5.1.1) onto the Nexus phone. This also led us away from further exploration of setting Tasker to be a System App.

5.3 Rooting the Device

Initially we tried giving super user access to the user on the phone using SuperSU, and we realized that we needed root access to grant user SU privileges. Initially it took some time to figure out how to correctly unlock the boot loader on our phone, since there were many websites online that provided solutions that were deprecated. Once we unlocked the bootloader, we attempted to root the phone with an outdated version of the TWRP utility, but this ended up crashing as soon as the phone was rebooted. After finding the most updated version of TWRP, we were able to save a safe version of the operating system state correctly. After all these changes, we were able to reboot with root access.

5.4 Broadcast Receiver Parameters Issue

As mentioned in 4.3 Run State, we had to resolve an issue related to the vectors being destroyed each time the broadcast receiver received the *Broadcastintent.ACTION_BATTERY_CHANGED* signal.

5.5 Building AOSP

A significant portion of this project was spent on trying to download, compile and run an OS built using the AOSP.

Initially, we tried to use a Mid 2012 MacBook Pro to build AOSP. This posed significant challenges. For example, we had to set up a case-sensitive drive to build the project on, as Mac's have case-insensitive file systems by default [4]. We resolved this by migrating build environments from an older MacBook Pro to a server-class desktop Linux machine.

Downloading the app was problematic as the repo is upwards of 20GB in size [5]. As the initial download had been started on our laptop, we often ran into issues such as running out of disk. We also faced issues such as wanting to pause the download process, as we had a class which required laptop use.

6. Results

We built a website [6] and have a GitHub URL [7] to download all the source code from to support users that wish to use our app. This website features a video demonstration [8] showing users how to engage with our app, and proving that it functions as we expect it to.

7. Future Work

We have identified a few areas for future work as listed below.

- Allow users to define their own battery threshold values.
- Explore options that enable operating system support to prevent certain apps from being rescheduled (dangerous)
- Improve performance of our app.
- Improve UI of app.
- Remove "widgets" from list of launchable apps.
- Conduct surveys to understand who uses our app and what features they would like to see added to the app.

8. Conclusion

We met all of the objectives we set out to achieve and thoroughly enjoyed the app building process. Our exploration eliminated ideas such as modifying the kernel, or using existing frameworks to achieve our objectives. Learning about the various complexities associated with developing an app was also very challenging. Some key areas for future work were identified, which will hopefully lead to the user adoption of the app.

Overall, this project can be concluded to be a success for the reasons mentioned above. [3]

9. Works Cited

[1] "Tasker," [Online]. Available: http://tasker.dinglisch.net/.

[2] Ashish, "GuidingTech," [Online]. Available: http://www.guidingtech.com/45575/tasker-alternatives/.

[3] Google, "AOSP - Stock Android Images," [Online]. Available: https://developers.google.com/android/images.

[4] Google, "AOSP - Building," [Online]. Available: https://source.android.com/source/building.html.

[5] Hagen, "Google Groups Discussion - Size of AOSP repo," [Online]. Available: https://groups.google.com/forum/#!topic/android-building/9tacBIL9YcM.

[6] P. B. a. S. Deo, "Energy-Aware Task Killer Main Site," [Online]. Available: https://sites.google.com/view/energy-aware-task-scheduler/.

[7] P. B. a. S. Deo, "Energy-Aware Task Killer GitHub," [Online]. Available: https://github.com/pban1993/energy_aware_task_killer.

[8] P. B. a. S. Deo, "Energy-Aware Task Killer YouTube Video," [Online]. Available: https://www.youtube.com/watch?v=UW26x_0frjk.

[9] Google, "AOSP - Running the Build," [Online]. Available: https://source.android.com/source/running.html.