Francesco Leone 82023140

NON-INVASIVE BATTERY SAVER FOR ANDROID SMARTPHONE

INTRODUCTION

In the last ten years the smartphones have become incredibly powerful. One can use the smartphone to entertain, to work, to use different tools such as calculator and GPS and many other purposes. One of the main advantages of the smartphones is the portability. Over time, the quality of the batteries installed in this kind of devices has significantly improved, allowing the smartphones to work at medium-high intensity even for a whole day before to shut down.

However, this could be not enough. Even the best battery ever may not suit your needs if you are not careful about the background work of the smartphone. To improve the efficiency of the battery, limiting the processes running in background is necessary. This aspect is so important that Android implements its own battery saver, in order to give to the user the possibility to reduce the unwanted background work of the smartphone and to make the battery last longer.

In this report I will present FLBatterySaver, an Android app that implements a new battery saver. Its aim is to extend the battery life in an effective though less invasive way compared to Android native battery saving mode. The basic concept is the same: to close unneeded background tasks that could drain power uselessly. However, rather than to reduce the potential of the device limiting every application and functionality, FLBatterySaver limits only those processes that cannot be seen by the user, thus it is invisible from the point of view of the user experience.

RELATED WORKS

Currently the main option to extend battery life on Android devices is the native battery saving mode that is included with the Operative System, introduced with Android 5.0 Lollipop. It is intended to be used to extend the battery life when it is almost dead already, because of its invasiveness. In particular, when the Android saving mode is on [1]:

- Apps do not run in the background
- Apps refresh their content only if opened
- Synchronization for apps such as email and news services is suspended
- Location services are stopped when the screen is turned off
- Notifications may be delayed
- Some visual and hardware effects (such as screen brightness and vibration) may be reduced or turned off
- The listening function “OK Google” is suspended

The invasiveness of this approach comes from the fact that each of these points affects the general performance of the device, so the smartphone is slower, the apps take longer to be launched and some features cannot be used without switching off the battery optimization. This is also the reason why, even if smartphones often allow to set a personalized threshold, the default threshold for battery percentage at which the save mode automatically starts working is 20%. A higher threshold would worsen the user experience by limiting the smartphone potential for long periods.

To compensate this low threshold, Android implements also an “Extreme Battery Sever” option. When this option is turned on, every feature is suspended except for calls and messages services and the battery life is stretched as possible. This option is extremely useful to avoid the smartphone to die when the battery is very low. In this way, the user can use the full potential of the smartphone until it reaches the 20% of battery, then he accepts some compromises between performance and energy saving, and eventually with the Extreme Battery Saver mode he can use at least the most basic functionalities of the mobile phone until it dies.

PROPOSAL

1.SAVE MODE

FLBatterySaver extends the battery life in a non-invasive way by limiting background processes, but with some differences with respect to Android battery saver. In fact, the limitation involves only application that are not OS-related and have not any foreground service. Firstly, this ensures that the smartphone will continue to work properly when FLBatterySaver’s save mode is on, since no application that is required to run the OS will be affected.

Secondly, since no foreground service is closed by FLBatterySaver, the user experience will be not compromised: if an application is currently being used by the user, if it launches a background process to update its data or do anything else, this will not be closed and so the application will still run smoothly. As a consequence, notifications mechanism is not influenced by FLBatterySaver’s save mode, because every time an app sends a notification on the screen, this process is considered to be in foreground by Android.

In the first implementation of FLBatterySaver, the application was also supposed to automatically switch off the Wi-Fi, the location services and the Bluetooth, but in the end this feature was removed. The reason is that, on recent smartphones, a third-party application needs a runtime permission every time it wants to switch off these services. Thus, on recent devices this feature would have been annoying rather than useful.

The intention of FLBatterySaver is different from Android native battery saver’s one: while Android’s battery saver aims to extend the battery duration as much as it can in the last span of its life, FLBatterySaver intention is to extend this duration in a softer way but working for a longer period. The reason why non-invasiveness has been preferred to effectiveness is that no third-party application could ever be as effective as Android native battery saver, because it would lack the system permissions to limit background processes without constraints. Hence, it is more useful to implement a different approach and give a different possibility to the user, rather than to implement the same approach in a less effective way.

2.THRESHOLD

Another feature of FLBatterySaver is the battery threshold: the user can set a threshold for the percentage of battery power at which the save mode will automatically start working, when it is enabled. In this way, the background processes will not be influenced at all when the smartphone has enough power according to the user. When enabled, the save mode will automatically suspend its work if the battery percentage is over the threshold. The save mode's work is suspended when the smartphone is charging, too. This is for the same reason of the threshold: FLBatterySaver wants to be as less invasive as possible. So, as long as there is no need to save the battery, the application will stop working.

3.GUI

For what regards the user interface, FLBatterySaver's UI is formed by a single main page that includes anything is needed. On the screen, the user can immediately find some basic information about the battery, that is the percentage of battery charge remained, what is the status of the battery and what is its temperature. The status can have seven different values: good, dead, cold, overheat, overvoltage, unknown, and unspecified failure. Both the temperature and the status values are taken from the OS, so the data is reliable.

From the main page the user can also set the threshold value for the battery percentage and can activate the save mode through a switch. The save mode does not work if the switch is set to off. Beneath the save mode switch there is an info button. By tapping on it, the user is shown a dialog box that explains briefly how the application works, what the threshold does, what is the default threshold, and that the save mode is not enable if the smartphone is charging.

IMPLEMENTATION

1.GUI and BATTERY INFO

FLBatterySaver has been developed using Android Studio (version 4.1.2). It is the official IDE for Android development, powered by IntelliJ, and so it is well supported, has always updated official documentation and provides an Android emulator to safely test the application on different kind of devices. The used programming language is Java. It was used to implement the background work of the application and the dynamic behavior of the UI. For the static representation of the UI, XML was instead used.

XML is a markup language supported by Android Studio that allows the developer to implement a graphic interface for an application in a simple and direct way: every kind of Android component (such as switches, buttons, image boxes) is simply declared and positioned on the screen. Android Studio also provides a preview window, in order to have a general overview on how the screen will look and what should be corrected.

With XML, Android Studio can also automatically create listeners for components that must detect touch inputs. However, the behavior of the application when an input is detected has to be controlled with Java code. This is particularly useful for components that have a simple behavior. In FLBatterySaver automatic listeners were used for the save mode switch and the info button. In the first case the application check if the touch enabled or disabled the switch and, if needed, launches the save mode's process. In the second case, it simply launches the dialog box that explains how the application works.

For the threshold input box, the listener was manually created with Java because it has a more complex behavior. The application has to handle two cases:

- the user changes the threshold without confirming the choice; in this case the application does not do nothing
- the user changes the threshold and confirms the choice; in this case the application saves the new threshold internally and, if the save mode is enabled, modifies this parameter in the save mode's process. Resetting the default threshold is included in this case

When FLBatterySaver is closed, it saves the current threshold and the current state of the switch, so that the UI will show the correct values as it will be opened again. In order to do this, the SharedPreferences class [2] was used. This is an XML file that Android can automatically create for each application and can be used to save a small collection of simple data (such as usernames, passwords, numbers) in a key-value map, with basically no impact on the storage.

To retrieve the information about the battery shown on the screen, the IntentFilter class was used [3-4]: this is an Android class that can be used to detect a specific kind of message among the ones that are broadcasted by the OS. FLBatterySaver detects the ACTION_BATTERY_CHANGED messages, that contain all the necessary information (percentage, status, temperature). It was not possible to obtain more information, because smartphones normally do not have a specific sensor for battery. For this reason, even if Android has different APIs that can be used to know data such as residual charge (in µAh), these can be used only on some devices, such as the newest Nexus models, that have an ad hoc sensor for the battery.

2.SAVE MODE

The save mode implementation is based on the Worker class [5]. This Android class is intended to be used for deferrable, asynchronous simple tasks, called "works", that should survive even if the application that launched them is closed or the smartphone is restarted. Furthermore, Android OS automatically schedules and handles the works in a way that optimizes the battery usage as much as possible. For these reasons, this was the best implementation choice for the application.

Among the features of the Worker class, there is the possibility to implement two kinds of scheduling: one-time and periodic works. For FLBatterySaver the second option was used. Every 15 seconds, a Worker instance checks the list of installed applications and then kills every killable background process. The same instance will repeat its task after 15 minutes. So 60 instances of the Worker class are used in total. A background process is deemed as killable if it is not OS-related (processes that are used to run the Operative System will not be stopped) and is not related to any application in the foreground (processes that serve an application currently used by the user will not be stopped, so the user experience will not be affected).

To kill the background processes, the Worker class takes the following steps:

1) A list of all the installed applications is retrieved with the getInstalledPackages() method from the PackageManager class [6].
2) For each application, the killBackgroundProcesses() method from ActivityManager class [7] is used to kill its processes if there are any and if it is possible

The killBackgroundProcesses() method perfectly fits the purpose for two reasons. Firstly, it allows to close the processes of other applications without special system permissions. Secondly, when this method is used, the OS automatically decides what processes can be closed and what cannot, basing on the rule mentioned before. Thus, the Worker does not need to control itself which processes must be killed and possible anomalous behaviors of the smartphone, caused by having killed a non-killable process by accident, are avoided.

EVALUATION

FLBatterySaver was tested in two different environments: Android Studio's internal emulator for stability and compatibility, and a real smartphone for effectiveness. To test the stability of the application, its behavior in different situations (e.g., when the application is switched with another one or suddenly closed) and with different hardware inputs (e.g., when the smartphone is rotated or the screen is turned off) was analyzed. The application has never crashed, and the different GUI components always showed the correct information. Thus, the application has been deemed stable.

The emulator was also used to test the compatibility of the application with different brands of smartphones. For this purpose, three different devices were emulated: a Google Pixel 3a, a Nexus 5 and a Nexus 6. The application worked correctly on all the three devices. The GUI never exhibited any kind of graphic distortion and the different GUI components were always well aligned, as expected by the design.

For what regards the effectiveness of the save mode, a Huawei P20 lite was used as a real test device. The metric used to check the load of background processes on the system was the amount of RAM used by the applications. Thanks to the developer options available on Android, no external tools were needed to extrapolate data about the RAM usage. The measurements were taken in three different condition:

- Before installing the application: this measurement was used to compare the impact of FLBatterySaver on the system. The amount of RAM used by the applications detected by the system was about 700MB on average.

- After installing FLBatterySaver, with the save mode off: this measurement is useful to check if the application has a negative impact on the RAM, and thus on the battery, nullifying the advantages of the save mode. The amount of RAM used in these conditions was again 700MB on average. Hence, the application not only does not overload the RAM, but is in fact almost invisible from this point of view.
- After turning the save mode on: in these conditions, the amount of RAM used by applications on average was only about 400MB. FLBatterySaver's save mode proved to be very effective, since it was able to reduce by about 40% the amount of RAM used by background processes on the test device.

The results can be considered a bit conservative. Since on the smartphone used for testing there were not too many installed applications, the work for the save mode was not heavy. Probably, the save mode would be even more effective if running on a smartphone that is used more intensively and with more installed applications.

CONCLUSION

Improving the efficiency of the battery is fundamental to have a smartphone that is not going to die when we need it. To achieve this goal, Android has implemented its own battery saver. However, this is really invasive, since drastically reduces the performance of the smartphone. For this reason, it is normally used only when the battery of the smartphone is already hopelessly low.

FLBatterySaver is a good alternative to Android native battery saver to extend the battery life in a non-invasive way. Its purpose is to save a decent amount of battery charge over a long period, rather than to save the highest amount possible in a short period.

The results show that the application has a positive effect in limiting the background processes, since about the 40% of the background processes were closed by FLBatterySaver's save mode on the test device. It has also proved to be almost invisible to the user with respect to Android's battery saver, and so can be used for much longer periods without negative effects on the performance.

There are at least two ways to enhance FLBatterySaver, and these will be the subject of future work. Firstly, the effectiveness of FLBatterySaver's save mode over long periods of usage should be tested. For this report, the only metric used to confirm the effectiveness of the save mode was the RAM usage, that is a good indicator on how many processes are running in background but may have less importance than expected for the overall duration of the battery.

Secondly, further tests should be run to find the optimal number of Worker class instances to implement the save mode. Currently, the application uses 60 instances, that is the number required to clean the RAM from the background processes every 15 seconds. With more instances the effectiveness of the save mode would be increased, whereas with less instances the load of the application on the system, that is already low, can be further reduced. Therefore, the optimal tradeoff should be found.

References

[1] https://support.google.com/pixelphone/answer/6187458?hl=en#zippy=

[2] https://developer.android.com/reference/android/content/SharedPreferences

[3] https://developer.android.com/training/monitoring-device-state/battery-monitoring#java

[4] https://developer.android.com/reference/android/os/BatteryManager

[5] https://developer.android.com/topic/libraries/architecture/workmanager/how-to/managing-work

[6] https://developer.android.com/reference/android/content/pm/PackageManager

[7] https://developer.android.com/reference/android/app/ActivityManager