

CSC714 – Final Report

Fall 2011

An Application Profiler for Android

<http://www4.ncsu.edu/~arezaei2/CSC714/>

Arash Rezaei
arezaei2@ncsu.edu

Gopikannan Venugopalsamy
gvenugo@ncsu.edu

Introduction

With the advent of smart phones, mobile computing has been revolutionized and many new opportunities have been introduced into the world of research. A challenge with today's smart phones is their security issue with regards to third party applications. The way a given malicious application uses the phone resources like CPU, battery, network and private data is different from the others. The Android market is getting to a considerably large share and screening that for malicious applications becoming so hard, if not impossible. In this project, our aim is to provide an application profiler for Android platform. This can be achieved by implementing a profiling daemon which gathers the formation related to the usage of system resources like computing, memory, file manipulation, network and other sensing resources/information related to each application.

Tasks:

- Install Eclipse + SDK (both)
- Background reading - profiler applications (both)
- Find the classes to get the monitoring data (both)
- Daemon, monitoring: Network, files (creation/deletion/change) (**Arash**)
- Monitoring: CPU, Battery, Sensors, Memory (**Gopi**)
- Design the profiler (both)
- Profiler Implementation (both)
- Testing and Final Report Submission (both)

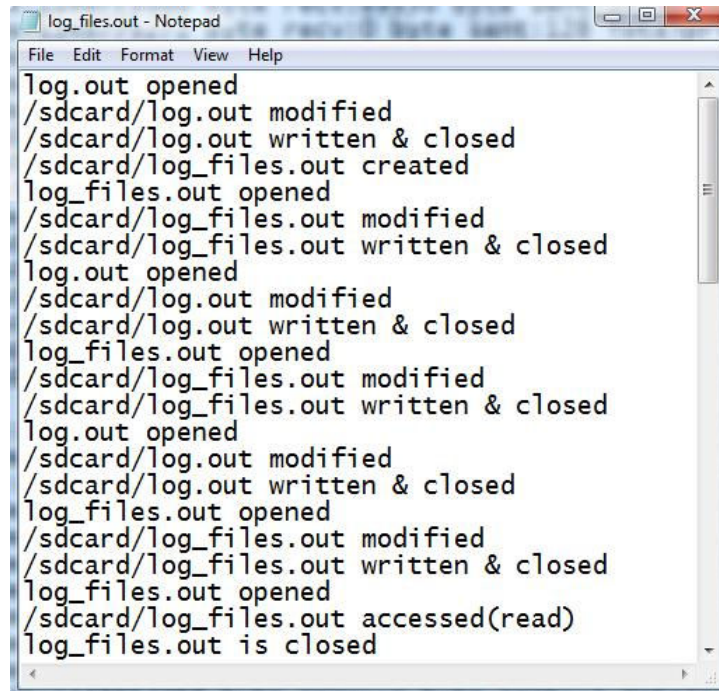
Design & Implementation

1. **Monitoring Service:** A Service is an application component either performing a longer-running operation while not interacting with the user or to supply functionality for other applications to use. Services can be started with *Context.startService()* and *Context.bindService()*. So services are run in the background. We implemented two monitoring service. The first one is a periodic process that wakes up every T seconds and gathers information about all applications currently running in the Android device and dumps the data into a log file. The second service is an event-based file system monitoring service which logs every changes in the file system.

2. **Memory Usage:** Memory in Android is shared across multiple processes. *ActivityManager* class has a method *getMemoryInfo()* which returns overall memory usage of the device. This is usually helpful to find out when there is no more memory to create new background processes. The *getProcessMemoryInfo()* method can be used to obtain the memory usage of a process running in device. This returns a memory info object that has various components *Pss*, *PrivateDirty* and *sharedDirty*. The *Pss* metric will help us to get an idea about the relative RAM usage of processes. *PrivateDirty* is the number of Dirty pages belonging to the process which cannot be flushed to disk. All these components can be collectively used to profile the memory usage of different applications.
3. **Sensor Power Usage:** Sensor manager is the service that handles all kinds of sensors in device. *Batterystats.Uid.sensor* is a package that can be used to obtain statistics of sensor usage of a process. It can be used to obtain the type of sensor used by application and the duration of usage. The *com.android.internal.os.PowerProfile* package can be used to obtain the power consumed when that particular sensor is on. The energy consumed by application's sensor usage can be found by multiplying the returned power consumption factor with duration of usage.
4. **CPU Power Usage:** The *Activity Manager* can be used to obtain the *Uid* of the running processes. For each running process CPU usage can be found using the package "*android.os.BatteryStats.proc*" can be used along the *Uid* of the process to get the user *time*, *system* time and foreground time of the application. The sum of these can be multiplied along the power factor obtained from "*com.android.internal.os.PowerProfile*" package which gives us the CPU usage of that application.
5. **File manipulation:** Monitoring files (creation/ deletion/ change) can be done using the *FileObserver*, which fires an event after files are accessed or changed by any process. Each *FileObserver* instance monitors a single file or directory. If a directory is monitored, events will be triggered for all files and subdirectories (recursively) inside the monitored directory. An event mask can be used to specify which changes or actions to report.
6. **Logging Mechanism:** The monitoring service writes out the gathered statistics to a log file in SD card as the log file keeps on growing and the internal memory in device will not be enough. The gathered data is written to a log file at the beginning of each period along with timestamp. The changes in the file system are recorded in a separate log file. Instead of writing it to the log file at every single event. To improve the I/O performance and avoid writing to the log after each file-related event, we aggregate the writes to log file and perform the write once in each period.

The second option in our system is that these data can be send to a server for processing. So we also write the data to a socket.

Snapshots:



```
log_files.out - Notepad
File Edit Format View Help
log.out opened
/sdcard/log.out modified
/sdcard/log.out written & closed
/sdcard/log_files.out created
log_files.out opened
/sdcard/log_files.out modified
/sdcard/log_files.out written & closed
log.out opened
/sdcard/log.out modified
/sdcard/log.out written & closed
log_files.out opened
/sdcard/log_files.out modified
/sdcard/log_files.out written & closed
log.out opened
/sdcard/log.out modified
/sdcard/log.out written & closed
log_files.out opened
/sdcard/log_files.out modified
/sdcard/log_files.out written & closed
log_files.out opened
/sdcard/log_files.out accessed(read)
log_files.out is closed
```

Fig 1: Snapshot of the log file recording file system events

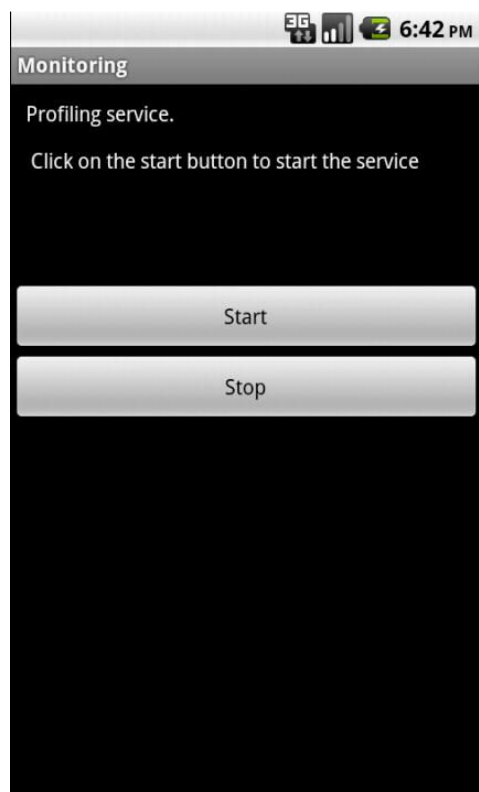


Fig 2: Profiler Application

Test Cases

Test cases	Result
Running different applications simultaneously and check whether profiler logs the information of running activities.	Passed
Make an application to read and write a file and check whether the events are logged.	Passed

Future work:

- Calculate the cost of sending/receiving a byte and use it for calculating total power consumption.
- Gain root access and get detailed information of applications like tracing system calls, file manipulation by each process (/proc/ID/fd).
- Analyze the gathered data and predict the behavior of various applications.

References:

- [Service] <http://developer.android.com/reference/android/app/Service.html>
- [Memory Usage]
- <http://stackoverflow.com/questions/2298208/how-to-discover-memory-usage-of-my-application-in-android>
- [Memory Usage] [http://developer.android.com/reference/android/os/Debug.html#getMemoryInfo\(android.os.Debug.MemoryInfo\)](http://developer.android.com/reference/android/os/Debug.html#getMemoryInfo(android.os.Debug.MemoryInfo))
- [Sensor Usage] <http://hi-android.info/src/com/android/internal/os/PowerProfile.java.html>
- [Sensor Usage] http://www.androidjavadoc.com/1.1_r1_src/android/os/BatteryStats.Uid.Sensor.html
- [CPU usage] <http://hi-android.info/src/com/android/internal/os/PowerProfile.java.html>
- [File Manipulation] <http://developer.android.com/reference/android/os/FileObserver.html>