CSC714 – Progress Report

Fall 2011

An Application Profiler for Android

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

Arash Rezaei
arezaei2@ncsu.edu

Gopikannan Venugopalsamy
gvenugo@ncsu.edu

Problem Statement

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.

Design

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. Our monitoring service 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 as following.

```
{Time_stamp}, {App1}
CPU:{value},Memory:{value},Sensores:{value},openfiles:{value},updatedfiles:{value},
deletedfile:{value}, newfiles:{value}
{Time_stamp}, {App2},
CPU:{value},Memory:{value},Sensores:{value},openfiles:{value},updatedfiles:{value},
deletedfile:{value},newfiles:{value}
..
```

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 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 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.

Milestone:

Task	***Status***
Install Eclipse + SDK (both - due **OCT 27**)	completed
Background reading - profiler applications (both - due **OCT 30**)	completed
Find the classes to get the monitoring data Daemon, monitoring: ~~Network~~, files (creation/deletion/change) (**Arash**) Monitoring: CPU, ~~Battery~~, Memory, Sensors (**Gopi**)	completed
Design the profiler(due **NOV 10**)	completed
Profiler Implementation (due **NOV 23**)	
Analysis of gathered data (due **NOV 25**)	
Final project report (due **NOV 29**)	

Update: During this project, we found out that Network usage and Battery usage are way more complex than the given time we had, so we don't have a solution for them so far. Instead we looked into memory usage. Another thing that we could not find a way to get information is performance counters, mainly because of the limitations exposed by Android.

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
- [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