

블로그 검색

블로그 플래킹

다음 블로그»

블로그 만들기 | 로그인

# ENEAA *android blog*

Wednesday, July 1, 2009

## The System Server in Android

In this post I will add some more detail on the system server in Android. The system server is the core of the Android system and as described in the boot sequence post it is started as soon as Dalvik is initialized and running. The other system services will be running in the context of the System Server process. We will start by looking at the code that runs when the System Server starts. This code is found in the file `frameworks/base/services/java/com/android/server/SystemServer.java` (in the open source project tree) and we will start this discussion from the main entry point.

```
/**
 * This method is called from Zygote to initialize the system. This will
 * cause the native
 * services (SurfaceFlinger, AudioFlinger, etc..) to be started. After
 * that it will call back
 * up into init2() to start the Android services.
 */
native public static void init1(String[] args);
public static void main(String[] args) {
    // The system server has to run all of the time, so it needs to be
    // as efficient as possible with its memory usage.
    VMRuntime.getRuntime().setTargetHeapUtilization(0.8f);

    System.loadLibrary("android_servers");
    init1(args);
}
public static final void init2() {
    Log.i(TAG, "Entered the Android system server!");
    Thread thr = new ServerThread();
    thr.setName("android.server.ServerThread");
    thr.start();
}
```

The first thing that happens is that the server will load a native library called `android_server` that provides interfaces to native functionality. Source files for this lib are placed in `frameworks/base/services/jni/`. Then the native init method that will setup native services is called, `init1(args)`, and executed. The name of the function that implements this is `system_init` and it resides in `frameworks/base/cmds/system_server/library/system_init.cpp`. After setting up the native services there is a callback:

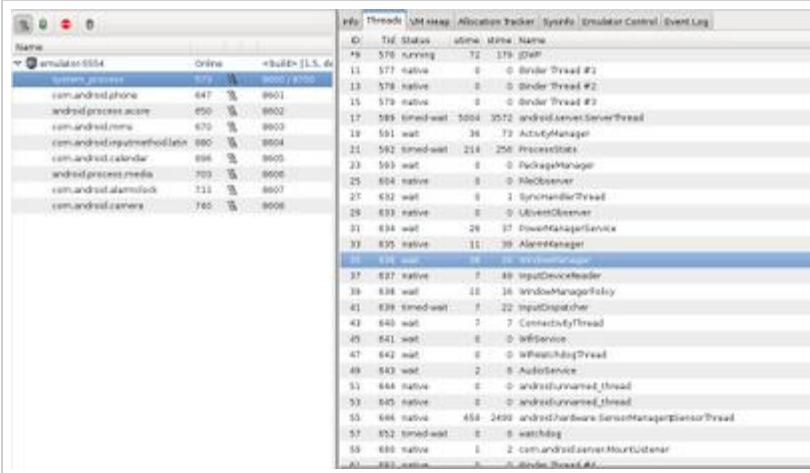
```
runtime->callStatic("com/android/server/SystemServer", "init2");
```

to `init2()` above to create the server thread. This thread will start the remaining services in system according to the necessary start order. A snippet of the initial sequence gives:

```
// Critical services...
try {
    Log.i(TAG, "Starting Power Manager.");
    power = new PowerManagerService();
    ServiceManager.addService(Context.POWER_SERVICE, power);
    Log.i(TAG, "Starting Activity Manager.");
    context = ActivityManagerService.main(factoryTest);
    Log.i(TAG, "Starting telephony registry");
    ServiceManager.addService("telephony.registry", new
TelephonyRegistry(context));
    AttributeCache.init(context);
    Log.i(TAG, "Starting Package Manager.");
    pm = PackageManagerService.main(context,
        factoryTest != SystemServer.FACTORY_TEST_OFF);
    ActivityManagerService.setSystemProcess();
    mContentResolver = context.getContentResolver();
    Log.i(TAG, "Starting Content Manager.");
    ContentService.main(context,
        factoryTest == SystemServer.FACTORY_TEST_LOW_LEVEL);
    Log.i(TAG, "Starting System Content Providers.");
    ActivityManagerService.installSystemProviders();
    Log.i(TAG, "Starting Battery Service.");
    BatteryService battery = new BatteryService(context);
    ServiceManager.addService("battery", battery);
    Log.i(TAG, "Starting Hardware Service.");
    hardware = new HardwareService(context);
    ServiceManager.addService("hardware", hardware);
    // only initialize the power service after we have started the
    // hardware service, content providers and the battery service.
    power.init(context, hardware, ActivityManagerService.getDefault
battery);

    Log.i(TAG, "Starting Alarm Manager.");
    AlarmManagerService alarm = new AlarmManagerService(context);
    ServiceManager.addService(Context.ALARM_SERVICE, alarm);
    ...
}
```

We see that the power manager is started first, followed by the activity manager and the other services. There are a lot more services started after these initial and if you are interested to look in the `SystemServer.java` file. Each service is running in a separate Dalvik thread in the `SystemServer` process. To give some info on the components making up the system server you may have look at it using the DDMS tool:



ID	Tid	Status	utime	stime	Name
*8	576	running	72	179	[DPM]
11	577	native	0	0	Binder Thread #2
13	578	native	0	0	Binder Thread #2
15	579	native	0	0	Binder Thread #3
17	589	timed-wait	3004	3572	android.server.ServerThread
19	591	wait	36	73	ActivityManager
21	592	timed-wait	218	256	ProcessState
23	593	wait	0	0	PackageManager
25	604	native	0	0	RelObserver
27	632	wait	0	1	SynchronizerThread
29	633	native	0	0	UiEventObserver
31	634	wait	26	37	PowerManagerService
33	635	native	11	39	AlarmManager
35	636	wait	36	26	WindowManager
37	627	native	7	49	InputDeviceReader
39	636	wait	10	16	WindowManagerPolicy
41	639	timed-wait	7	22	InputDispatcher
43	640	wait	7	7	ConnectivityThread
45	641	wait	0	0	WifiService
47	642	wait	0	0	WifiP2PServiceThread
49	643	wait	2	6	AudioService
51	644	native	0	0	android.unnamed_thread
53	645	native	0	0	android.unnamed_thread
55	646	native	654	2490	android.hardware.SensorManagerSensorThread
57	652	timed-wait	0	0	watchdog
59	680	native	1	2	com.android.server.MountController
61	680	native	0	0	Binder Thread #4

We see that the main Android services such as the activity manager, package manager, alarm manager etc. are running in their separate threads but as parts of the system server process:

/Mattias

Posted by Enea Android Team at [4:41 PM](#)

Labels: [Android](#)

**0 comments:**

**Post a Comment**

Comment as: Select profile...

**Post Comment**

**Preview**

[Home](#)

[Old](#)

Subscribe to: [Post Comments \(Atom\)](#)

---