SONY

Improving Android Bootup Time

Tim Bird
Sony Network Entertainment, Inc.
<tim.bird (at) am.sony.com>

Overview

- Android boot sequence
- Measuring bootup time
- Problem areas
  - Including some gory details
- Ideas for improvements
- Conclusions
- Resources

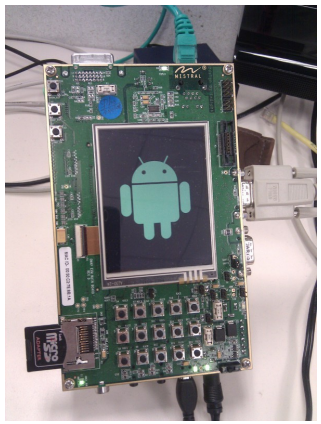
Android Boot Sequence

- Bootloader
- Kernel
- Init
  - Loads several daemons and services, including zygote
  - See /init.rc and /init.<platform>.rc
- Zygote
  - Preloads classes
  - Starts package manager
- Service manager
  - Starts services

Measuring boot time

- Systems Measured
  - Android Developer Phone (ADP1)
    - Qualcomm MSM7201 at 528 MHz with 192M RAM
    - Running Donut (1.6)

  - Nexus 1
    - Qualcomm 8250 at 1 GHz with 512 RAM
    - Running Eclair (2.1)
  - OMAP Evaluation Module
    - TI OMAP 3530 at 600 MHz with 128M RAM
    - Running Eclair
    - !! Using NFS-mounted root filesystem !!



Tools used

- Stopwatch
  - It's kind of sad that it takes so long that you can use a stopwatch
- Message loggers
  - Grabserial
  - Printk times
  - Android system log
- Bootchart
- Strace
- Dalvik method tracer*
- Ftrace*

Grabserial

- Tool for measuring time of printouts on a serial port, from a host machine
  - Only useful with EVM board, which has serial console
- Shows timestamp for each line received over serial console
- See http://elinux.org/Grabserial

Printk-times

- Kernel option for adding time stamp to each printk
  - Set CONFIG_PRINTK_TIME=y
  - Option is on "Kernel hacking" menu, "Show timing information on printks"
- Can save view on serial console, or after boot with ‘dmesg’
- Can turn on 'initcall_debug' on kernel command line
- See http://elinux.org/Printk_Times
- Init program also outputs to /dev/kmsg
  - Can adjust loglevel of 'init' program
    - Change “loglevel 3” to “loglevel 7” in /init.rc

Bootchart

- 'init' gathers data on startup
  - Must re-compile 'init' with support for bootchart data collection
- A tool on the host produces a nice graphic
- See http://elinux.org/Bootchart and http://elinux.org/Using_Bootchart_on_Android

Strace

- Shows system calls for a process (or set of processes)
- Is part of AOSP since Eclair
- Can add to init.rc to trace initialization.
  - For example, to trace zygote startup, in /init.rc change:

```
service zygote /system/bin/app_process -Xzygote /system/bin --zygote --start-system-server
```

to

```
service zygote /system/xbin/strace -tt -o/data/boot.strace /system/bin/app_process -Xzygote
    /system/bin --zygote --start-system-server
```

Android system log

- Android has built-in logging system
- Use ‘logcat’ command to view messages
- I added extra instrumentation for class preloading and package scanning
- I also set PARSE_CHATTY flag for package scanning
- I built my own tool (logdelta) to scan log and produce 'delta' times

Dalvik method tracer

- Method tracer is built into Dalvik
- Can be activated with ddms or using calls inside source
- Unfortunately, it didn’t work on the EVM platform, due to a problem with the clock
- Would be problematical for boot anyway, since thread of execution goes outside of Java (into C++ and kernel) for lots of important operations

Ftrace

- I could have really used ftrace for some things
  - Like to see page faults intermingled with system calls
- Kernel version for EVM (2.6.29) didn't support it
- Should be usable in future version of Android (Froyo is at 2.6.32)
- NOTE: ARM is missing function graph tracing
  - But there's a fix
  - See http://elinux.org/Ftrace_Function_Graph_ARM

Measurement results

- Stopwatch
- Grabserial
- Printk times
- Strace
- Bootchart
- Logcat

Stopwatch results

Platform	Time to static splash screen	To animated splash	To home screen	Number of apps
ADP1	4	32	57	39
Nexus 1	3.5	20	36	79
EVM	17*	37	62	45

NOTES: All times in seconds.

***EMV used a development bootloader, with high overhead and delays**

Printk-times results

- ADP1 kernel boot time

```
[    6.276367] Freeing init memory: 116K
```

  - 6.2 second kernel boot

- EVM kernel boot time

```
[44935.943115] OMAP DMA hardware revision 4.0
….
[44942.164093] VFS: Mounted root (nfs filesystem) on device 0:12.
[44942.170196] Freeing init memory: 164K
```

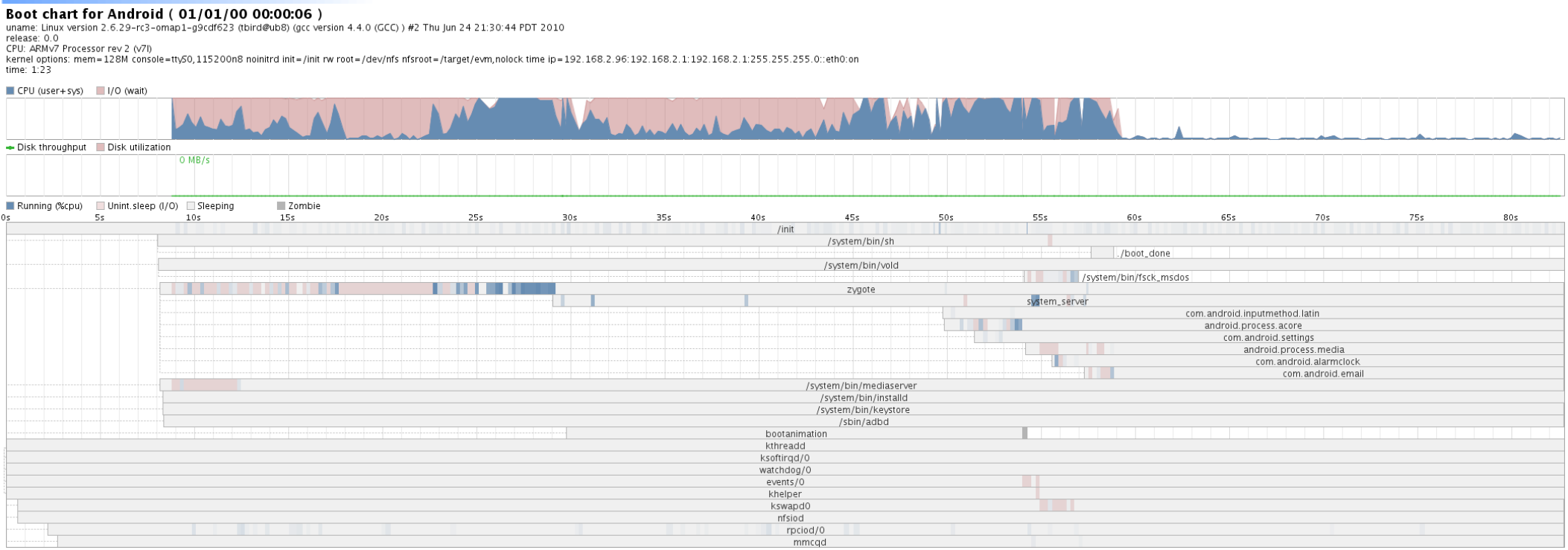
  - Clock not re-initialized at bootup
  - Have to diff end time with first reported time
  - 6.2 second kernel boot

Initcall_debug results

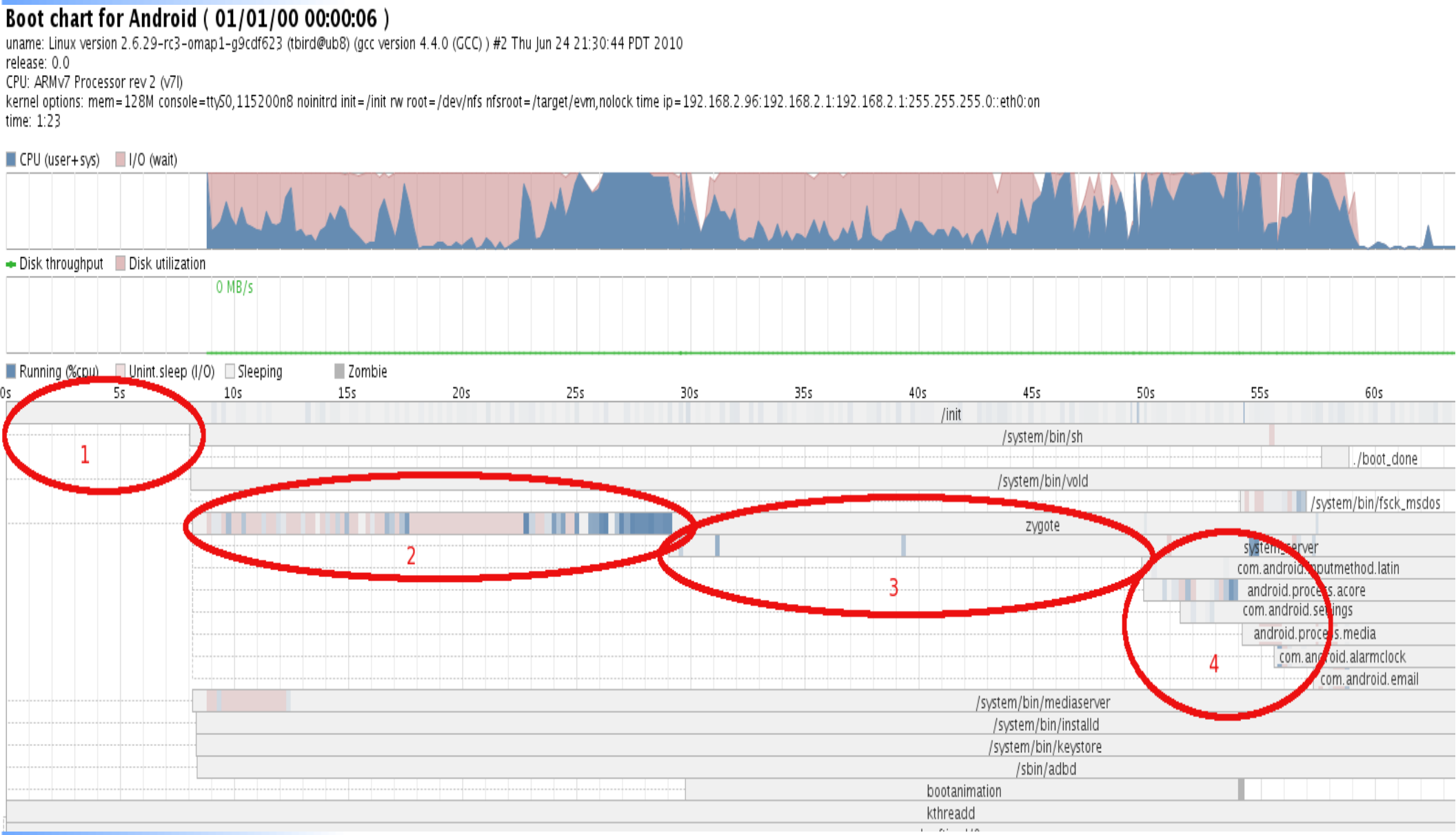
```
 77515 usecs - initcall omap_kp_init+0x0/0x1c returned 0
99152 usecs - initcall ehci_hcd_init+0x0/0xa4 returned 0
106334 usecs - initcall omap3evm_soc_init+0x0/0x9c returned 0
107079 usecs - initcall inet_init+0x0/0x1dc returned 0
115603 usecs - initcall omap_mmc_init+0x0/0x1c returned 0
148236 usecs - initcall omapfb_init+0x0/0x34 returned 0
154584 usecs - initcall omap_nand_init+0x0/0x30 returned 0
269770 usecs - initcall clk_disable_unused+0x0/0x8c returned 0
1514553 usecs - initcall ip_auto_config+0x0/0xdbc returned 0
```

- Interesting init_calls:
  - ip_auto_config = 1.51 seconds
  - ehci_hcd_init = .1 seconds
    - Usually more on real hardware
  - Various omap init routines = .86 seconds

Bootchart results



Bootchart closeup



Regions of interest

- Area 1 = kernel init
  - Approx. 8 seconds in this chart
- Area 2 = zygote class preloading
  - From approx. 8 to 29 seconds into boot
    - 21 seconds
- Area 3 = PackageManager package scanning
  - From approx 29 to 50 seconds into boot
    - 21 seconds
- Area 4 = System Services starting
  - From approx. 50 to 59 seconds into boot
  - 9 seconds

Logcat results

- ADP1:
  - preloaded 1514 classes in 11530ms
  - Time to scan packages: 10.064 seconds
- N1:
  - preloaded 1942 classes in 5360ms
  - Time to scan packages: 8.975 seconds
- EVM:
  - preloaded 1942 classes in 13619ms
  - Time to scan packages: 19.731 seconds

Problem Areas

- Bootloader init
- Kernel init
- Zygote class preloading
- Package scanning
- Service initialization

Bootloader init

- Outside scope of this talk
- Didn't measure commercial bootloader, only development one (U-boot)
- Is contained within first 4 seconds on phones (I think)

Kernel Init

- Is mostly the usual suspects
  - ip_auto_config
  - USB (ehci, ohci) init
  - OMAP flash and driver initialization
- See initcall_debug results
- Just follow standard instructions for optimizing initcalls in the kernel
  - See http://elinux.org/Boot_Time
- Some of this is board-specific
  - Depends on your flash controller
  - Is outside scope of this talk

Zygote class preloading

- Zygote pre-loads just under 2000 classes, and instantiates them in its heap
- Controlled by resource: preloaded-classes
- Which comes from source:
  - frameworks/base/preloaded-classes
- Google developers say you can adjust this as much as you like
  - Android can boot without preloading any classes
  - Can add or remove individual classes
    - I found that AutoText took 4 seconds to load
  - HOWERVER – it can result in bad application load times and memory usage later

Package manager package scan

- EVERY package is scanned at boot time
- Very deep nesting, with abstraction
- Not sure of exact set of purposes
  - But I see validation of certificates, permissions, capabilities and dependencies, etc.
- Very difficult to trace
  - It bounces between java, c++ and kernel
  - And uses mmaped files (meaning accesses cause page faults)!!
    - So it's not even using syscalls for reading the data

Package scan call tree

```
callstatic("com/android/server/SystemServer","init2")
 init2()
  main(context, factoryTest)
   PackageManagerService(context, factoryTest)
    scanDirLI(dir, flags, mode)
     scanPackageLI(file, file resfile, flags, mode)
      parsePackage(file, dest filename, metrics, flags)
       parsePackage(file, dest, metrics, flags)
        openXmlResourceParser(cookie, filename) (“AndroidManifest.xml“)
         openXmlBlockAsset(cookie, filename)
          openXmlAssetNative(cookie, filename)
           android_content_AssetManager_openXmlAssetNative
            openNonAsset(cookie, filename, flag)
             openNonAssetInPathLocked(filename, mode, asset_path)
              getZipFileLocked(asset_path)
               getZip(ap.path)
                get(path)
                 getZip()
                  SharedZip(path, modWhen)
                   ZipFileRO::open(path)
                    parseZipArchive(void)
                     get4LE(ptr)
                      (memory access)
```

parseZipArchive()

- Evil routine that builds an in-memory data structure for accessing a package file
- Scans the entire package, checking the content headers
  - Caused read of almost entire package
  - Touches every page in mmaped file, even if a sub-file in the archive won't be read later
  - i.e. Entire package file is read, when only the AndroidManifest.xml file is requested

Ideas for Improvements

- First, a side note on toothpaste..
- Kernel speedups
- Optimize package scan
- Optimize class preloading
- Miscellaneous optimizations
- Readahead??

Toothpaste

When you squeeze a tube of toothpaste, sometimes it just moves the toothpaste somewhere else in the tube, and nothing actually comes out.

- Same with optimizations:
  - Reduction in one area causes problem (speed or size) in some other area

Toothpaste (cont.)

- “Toothpaste effect” is demonstrated with class preloading and page cache effects
- I tried to improve things
  - But the I/O delays just moved somewhere else in the system
  - Sometimes making things worse
  - AutoText class preload example:
    - Eliminated preload of AutoText class and gained 4 seconds during class preloading, but /system/frameworks/frameworks-res.apk was just loaded later in the boot, costing 4 seconds there
  - Contacts.apk package scan example:
    - Moved AndroidManifest.xml to its own package, to avoid reading the entire (1.6M) package to build the package index, but next reference of Contacts.apk contents caused the index rebuild again (costing the entire page cache load)

Kernel speedups

- Outside the scope of this presentation
- See http://elinux.org/Boot_Time
- Should really be able to get kernel up in 1 second
  - Modulo network delays

Optimize Class Preloading

- Preload less, and let apps pay penalty for shared class and resource use
  - Move some classes to services, and have preloaded class be an accessor stub
  - Figure out how to share heap back with zygote
  - This needs a lot of analysis - Google Android devs know that the whole process of selecting what classes to preload is a black art
- Thread the heap construction
  - There is some evidence that class preloading has I/O waits, and would benefit from threading
  - Don't know if this is possible
  - NOTE: all threads need to terminate before spawning Android apps

Use pre-constructed dalvik heap

- Basic operation:
  - Snapshot the heap at end of preloading
  - Check for modifications to any class in preload list, and do regular preload
  - Otherwise, load pre-constructed heap
- Issues:
  - Don't know if this is possible
    - Need to find parts of heap that are identical on each boot
    - Probably need separate "always-the-same" and "may-change" classes
  - Need careful analysis, and might need knowledge of each class

Optimize Package Scan

- Most definitely! This should be first thing attacked
- I tried first order optimization
  - Removed per-file signature check in parseZipArchive()
  - This reduced duration of this routine (cumulative for 138 calls) by several seconds
  - But… total boot time was not reduced (!!)
    - Toothpaste effect strikes again!
- Need to continue analysis
  - May need to switch to a compressed flash file system, instead of managing indexing and compression in user space.

Miscellaneous

- zoneinfo inefficiences
  - Discovered with strace
  - Routine that does read syscall for 40 bytes, then 8 bytes, then another 8 bytes (hundreds of times)
    - No buffering at user level
    - Sloppy loop coding
  - Linear scan of timezone file
    - For a file not present!!
  - Probably only a few hundred milliseconds, but worth changing

readahead??

- One developer had an interesting result from just pre-filling the page cache
- Could use sreadahead to pre-fill page cache
- However, this just masks bad behavior
  - Contacts.apk (half of 1.6M) is read 4 times! during boot
  - Filling page cache makes reads after first one fast, but it would be better to avoid (most of) the reads altogether
    - Would be better to just optimize or eliminate parseZipArchive()
- sreadahead should be used dead last (after all other enhancements)

Conclusions

- Sorry - no speedups yet
- But, have a good foundation and set of tools for improving things going forward
  - Good idea of where time is spent

Observations

- “Premature optimization is the root of all evil”
  - Be very careful of optimizing wasteful operations
  - Better to improve or eliminate the operations, than hide the wasteful operations with caching
- Beware of systemic or architectural problems
  - Package management basically builds a persistent container and compression architecture in user space
  - Except, it does it poorly. (It rebuilds the in-memory data structure for indexing an archive over and over.)
  - Just use a file system, for heaven's sake!

Resources

- Wiki page for this talk: http://elinux.org/Improving_Android_Boot_Time
- Use android-porting, android-platform, and android-kernel mailing lists, depending on where your issue is
  - See

  http://elinux.org/Android_Web_Resources#Mailing_Lists
- My e-mail: tim.bird (at) am.sony.com

SONY

Thanks for your time

Questions and Answers