Improving Embedded Systems Boot time by Hibernation

Nicola La Gloria, Laura Nao
Kynetics
About Me

• I am a physicist
• Worked on autonomous navigation of LTA systems for planetary exploration
• Working on Android embedded systems since 2009
• CEO of Kynetics in Santa Clara, CA
• Passionate about retro computing
About Me

Atari 2600 “Heavy Duty”, Sunnyvale 1977
Agenda

- Android’s boot sequence
- Android Boot time optimizations
- Hibernation as boot optimization approach (SIM)
  - swsusp
  - Kernel Power Management
  - Drivers PM ops
- Single Image Mode on NXP on i.MX8MM
  - Hibernation on i.MX8MM
Android boot sequence

Boot time depends on the Android version and underlying H/W. It can vary from 30 to 40 sec.
Because Linux COW (lazy copy) regarding the number of apps running on the system, only one copy of the system classes is loaded in RAM (parent).

Picture: Karim Yaghmour
Optimizing Android (cold) Boot

- Start Zygote as early as possible
  - i.e. moving classes cache from /data to /cache
- Parallelize the Package Manager Service
  - Multiple threads for APK scanning.
- Split classes needed by the System (System Server) from those needed by the apps
- Prioritized System Services (i.e. to bring up the GUI sooner)
- 30% boot time reduction
- Decent amount of work on Android user space
Single Image Mode

- Embedded systems are designed around specific application(s) context with defined boundary conditions.
- Create a snapshot of the system from an "arbitrary" but consistent initial state.
- Boot the system by loading the same snapshot from the storage to memory.
Single Image Mode

Consistent Initial State

Create Single Image by Hibernation → Store image on swap → Power OFF

Power ON

Load image from swap → Restore image in memory

Consistent Final State
Power States

- **Power states:**
  - **Suspend-to-Idle:** this is all software. User-space frozen and low power I/O
  - **Standby:** non-boot CPUs are taken offline and all low-level system functions are suspended during transitions into this state
  - **Suspend-to-RAM:** everything in the system is put into a low-power state, except for memory, which should be placed into the self-refresh mode to retain its contents.
  - **Hibernation:** the kernel stops all system activity and creates a snapshot image of memory to be written into persistent storage (power off not mandatory in general).
Kernel Power Management

- Kernel **drivers** uses one or both of the following models:
  - **System sleep model**
    - Suspend to RAM
    - Suspend to Disk
  - **Runtime Sleep model**
    - Suspend-to Idle

This is something that device, bus, and class **drivers** collaborate on by implementing various role-specific suspend and resume methods to cleanly power down hardware and software subsystems, then reactivate them without loss of data.

- All phases involved in a particular model case, use PM domain, bus, type, class or driver **callbacks**, for example
  - dev->driver->pm = &ops <- dev_pm_ops
swsusp

- Hibernation stages:
  - Create an image
    - All suspend callbacks (prepare)
    - freeze* callbacks.
  - Save the image (we need to system to save the image!)
    - thaw* callbacks
  - Power off
    - poweroff* callbacks
Work your Drivers PM Ops.

- Some devices misconfigured in such a way to prevent suspend/resume callbacks during hibernation
- Use explicitly or implement `dev_pm_ops`
  - Alternatively, platform drivers’ `.suspend/ .resume` callbacks are used for hibernation thaw/freeze/restore/poweroff functionality
- Use helper macros like `SET_SYSTEM_SLEEP_PM_OPS` to fill in the thaw/freeze/restore/poweroff callbacks.
Hibernation Flow

Freeze →

cpu_suspend

cpu state → RAM

in_suspend = 1

swsuspsave

Thaw

swsusps_write

Restore

cpu_resume

sp

in_suspend == 1

nosave

Picture: Russell Dill
Restore Flow
Android User Space

• Ensure to take the system to a consistent and clean state before hibernation
  – removing unwanted wake-locks
  – force threads to release semaphores
• Repaint the GUI to remove any unwanted artifacts after resume
• Eventually check for pending Surface Flinger transactions
• Surface Flinger - HWC (HAL) display data sync. HWC component is tightly coupled to the kernel (GPU) and proprietary binaries (GLES, GPU drivers).
Code once, run just there

- Kernel (i.mx) and Android user space are tightly coupled in terms of versioning.
- This means that features that are at kernel level have to be reworked depending on the particular Android version.
- Some of the components may be proprietary (source).
- Some of the components are provided as binaries:
  - GPU user space drivers
  - GLES GP dependent libs
  - Video H/W acceleration
  - HWC HAL
  - ...
Single Image Mode on i.MX6

- Boundary Devices Nitrogen6x
  - NXP i.MX6Q
  - Vivante GC2000 GPU
  - SDCARD
  - 1 GB RAM
- Android 7.0, Kernel 4.9
- No image optimization (~114K pages)
- Not perfect conditions for hibernation/resume (I/O and pages allocation)
- Single Image is loaded after kernel initialization
- Requires memory allocation optimizations to create the image.
- Boot time ~12 seconds.
Single Image Mode on i.MX8MM

- Boundary Devices Nitrogen8M Mini
  - NXP i.MX8M Mini
  - Vivante GC Nano GPU
  - Boot from eMMC
  - 2 GB RAM
- Android P, Kernel 4.14
- No hibernate image optimization
- Single Image is loaded (125K pages) after kernel initialization
- Requires less memory allocation due to larger RAM optimizations to create the image.
- Boot time ~12 seconds.
Video available on: https://youtu.be/2JpS6uBnmtc
Integration of swsusp with U-Boot

- Restoring from hibernation is just about copying pages from disk into memory and jumping to an address, which U-Boot can do
- U-Boot has to be told what address to jump to
- U-Boot doesn’t know the contents or even location of the nosave pages
  - This requires a special cpu_resume function which carries the len, start and stop address of nosave memory.
  - cpu_resume function address is passed to u-boot by swap_info page
- Modification required in the kernel code
- Modification required in u-boot code
- Important contribution done by Russell Dill for OMAP (32bit)
Thank you!
Embedded Linux Conference
North America