Snapshot Booting on Embedded Linux

2011. 4.
Kang, Dongwook
What is snapshot booting?

- For fast boot-up time, restoring a system to a certain state with a snapshot image.
What We Did

- **2009**
  - Porting snapshot booting (swsus in linux-2.6.21) to HUINS-6410 (S3C6410)
  - NAND flash read with DMA (in kernel and u-boot)

- **2010**
  - Snapshot restoration in u-boot **

- **2011**
  - Supporting Android
  - Supporting SD cards in u-boot
Next Topics

- Inside of a Snapshot Image
- Shutdown / Boot-up Processes
- NAND Flash Read with DMA
- Snapshot Restoration in U-Boot
- When is Snapshot Booting Available?
Inside of A Snapshot Image

Processor + Device Controllers

S3C6410
ARM1176
ARM11 core
MMU
LCD
cache
UART
TCM
GPIO

Memory

+ snapshot

or

Processor + Device Controllers

Memory

or

Snapshot
Inside of A Snapshot Image

**boot_cmd**: resume=/dev/mtdblock3
resume_offset= 0x0

---

**swap table**

<table>
<thead>
<tr>
<th>index</th>
<th>swap dev Page #</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0xXXXXXXXXXXX</td>
</tr>
<tr>
<td>1</td>
<td>0xXXXXXXXXXXX</td>
</tr>
<tr>
<td>2</td>
<td>0xXXXXXXXXXXX</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>nr_meta_pages</td>
<td>0xXXXXXXXXXXX</td>
</tr>
<tr>
<td>nr_meta_pages+1</td>
<td>0xXXXXXXXXXXX</td>
</tr>
<tr>
<td>nr_meta_pages+2</td>
<td>0xXXXXXXXXXXX</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>nr_meta_pages + nr_copy_pages</td>
<td>0xXXXXXXXXXXX</td>
</tr>
</tbody>
</table>

---

**swsusp_info**

<table>
<thead>
<tr>
<th>index</th>
<th>page frame #</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0xXXXXXXXXXXX</td>
</tr>
<tr>
<td>1</td>
<td>0xXXXXXXXXXXX</td>
</tr>
<tr>
<td>2</td>
<td>0xXXXXXXXXXXX</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>nr_copy_pages - 1</td>
<td>0xXXXXXXXXXXX</td>
</tr>
</tbody>
</table>

---

**pfn table**

<table>
<thead>
<tr>
<th>index</th>
<th>page frame #</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0xXXXXXXXXXXX</td>
</tr>
<tr>
<td>1</td>
<td>0xXXXXXXXXXXX</td>
</tr>
<tr>
<td>2</td>
<td>0xXXXXXXXXXXX</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

---

**snapshot image**

nr_copy_pages * 4 kByte
Step 1. making snapshot image on memory

```
# echo disk > /sys/power/state

state_store() -> enter_state() -> pm_suspend_disk() ->
prepare_processes() // freeze processes
swsusp_shrink_memory() // shrink used memory
device_suspend() // suspend devices & save device controller regs
swsusp_suspend()
    local_irq_disable()
    swsusp_arch_suspend()
        save_arch_s3c6410() // save GPIO, VIC
        save_arch_arm11() // save cp15 registers
        save_arch_arm11core() // save arm core registers
        swsusp_save() // make snapshot image
            // (copy all used pages to free ones)
        restore_arch_arm11() // restore cp15 registers
        restore_arch_s3c6410() // restore GPIO, VIC
    local_irq_enable()
device_resume() // resume devices & restore device controller regs
```
Shutdown Process (2/2)

- Step 2. writing snapshot image to NAND flash

```c
pm_suspend_disk()
    ...
    swsuspd_write()
        snapshot_read_next() // init & write swsuspd_info
        get_swap_writer() // [ ]
        swap_write_page() // [ ]
        save_image() // [ ]
        flush_swap_writer() // write the last page of swap table
        mark_swapfiles() // write swsuspd_header
    power_down()
```
Step 1. restoring memory and jump

System is turned on

init() → software_resume()
  swsusp_check()
  prepare_processes()
  swsusp_read()
    snapshot_write_next()
  get_swap_reader()
  swap_read_page()
  load_image()
  device_suspend()
  swsusp_resume()
    local_irq_disable()
    swsusp_arch_resume()
      restore_arch_arm11core()

  // read & check swsusp_header
  // freeze processes
  // allocate memory for swsusp_info
  // read first page of swap table
  // read swsusp_info
  // read pfno table
  // read & restore snapshot image
  // copy colliding pages to safe pages
  // suspend devices
  // restore colliding pages
  // restore arm core registers
  // jump to swsusp_arch_suspend()
Boot-up Process

- **Step 2. restoring registers**

  (in swsusup_arch_suspend)
  - `restore_arch_arm11()` // restore cp15 registers
  - `restore_arch_s3c6410()` // restore GPIO, VIC

  (in swsusup_suspend)
  - `local_irq_enable`

  (in pm_suspend_disk)
  - `device_resume()` // resume devices & restore device controller regs
  - `unprepare_processes()` // wake up processes
NAND Flash Read with DMA

- **With DMA, its read performance improved**
  - 3.3 MByte/sec $\rightarrow$ 5.5 MByte/sec

- **This is meaningful**
  - Snapshot image loading time is the largest part
  - This time is decided by read speed and snapshot size
Experimental Results **
Experimental Results

- **HUINS-6410 Development Board**
  - S3c6410, 128M DDR, 128M SLC NAND
  - Linux-2.6.21, tinyX, matchbox
  - Snapshot image is 17MB

<table>
<thead>
<tr>
<th></th>
<th>Normal</th>
<th>Snapshot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot Loader</td>
<td>5.94</td>
<td>0.70</td>
</tr>
<tr>
<td>Kernel</td>
<td>8.29</td>
<td>0.78</td>
</tr>
<tr>
<td>Initial Script / Snapshot Loading</td>
<td>2.96</td>
<td>3.50</td>
</tr>
<tr>
<td>Application / Process &amp; Dev Resuming</td>
<td>6.09</td>
<td>0.21</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>23.28</td>
<td>5.19</td>
</tr>
</tbody>
</table>
Snapshot Restoration in U-Boot

- We can skip kernel loading and initializing

- We can load snapshot image faster **
  - No memory collision

- But, instead of kernel, u-boot need to initialize devices
  - timer, vectored interrupt controller, and other devices
Restoration Process in U-Boot

```
u-boot# bootss 4000000

do_bootss()
   read_swsusp_hdr()       // read swsusp_header
   copy_image             // restore snapshot image
   setup_peripherals()    // init clock, VIC, timer
   jump_to_resume()       // jump to restore_arch_arm11core()

restore_arch_arm11core()  // [ restore collided pages
               //   restore arm core registers
               //   jump to swsusp_arch_suspend() ]
```
Experimental Results
Huins 6410sys
- S3c6410, 128M DDR, 128M SLC NAND
- Linux-2.6.21, tinyX, matchbox,
- snapshot image size = 17MB

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot Loader</td>
<td>5.94</td>
<td>0.70</td>
<td>0.27</td>
</tr>
<tr>
<td>Kernel</td>
<td>8.29</td>
<td>0.78</td>
<td>0</td>
</tr>
<tr>
<td>Initial Script / Snapshot Loading</td>
<td>2.96</td>
<td>3.50</td>
<td>3.33</td>
</tr>
<tr>
<td>Application / Process &amp; Dev Resuming</td>
<td>6.09</td>
<td>0.21</td>
<td>0.23</td>
</tr>
<tr>
<td>Total</td>
<td>23.28</td>
<td>5.19</td>
<td>3.83</td>
</tr>
</tbody>
</table>
When is Snapshot Booting Available?

- **When slow shutdown is OK**
  - In aforementioned case, about 10 seconds increase

- **When free memory is enough**
  - About half free pages are required

- **When NAND flash is enough**
  - In Android, the snapshot can be over 200 MB

- **When snapshot image is not too big**
  - Its size decides the boot-up time
QnA
dkang@etri.re.kr

http://www.launchphotography.com/Antarctica_Penguins.html