A fast booting technology on embedded linux – mass production perspective

2008. 07.11

Hojoon Park
Embedded SW Technology Research Team

ETRI 한국전자통신연구원
Electronics and Telecommunications Research Institute

Table of Contents

- ❑ Fast Booting?
- ❑ An introduction to fast booting
  - ● Fast booting technology on PC
  - ● Fast booting technology on embedded system
- ❑ Useful fast booting technologies on embedded systems
  - ● Bootloader
  - ● kernel
  - ● Filesystem
  - ● Initial Script
  - ● Shared Library

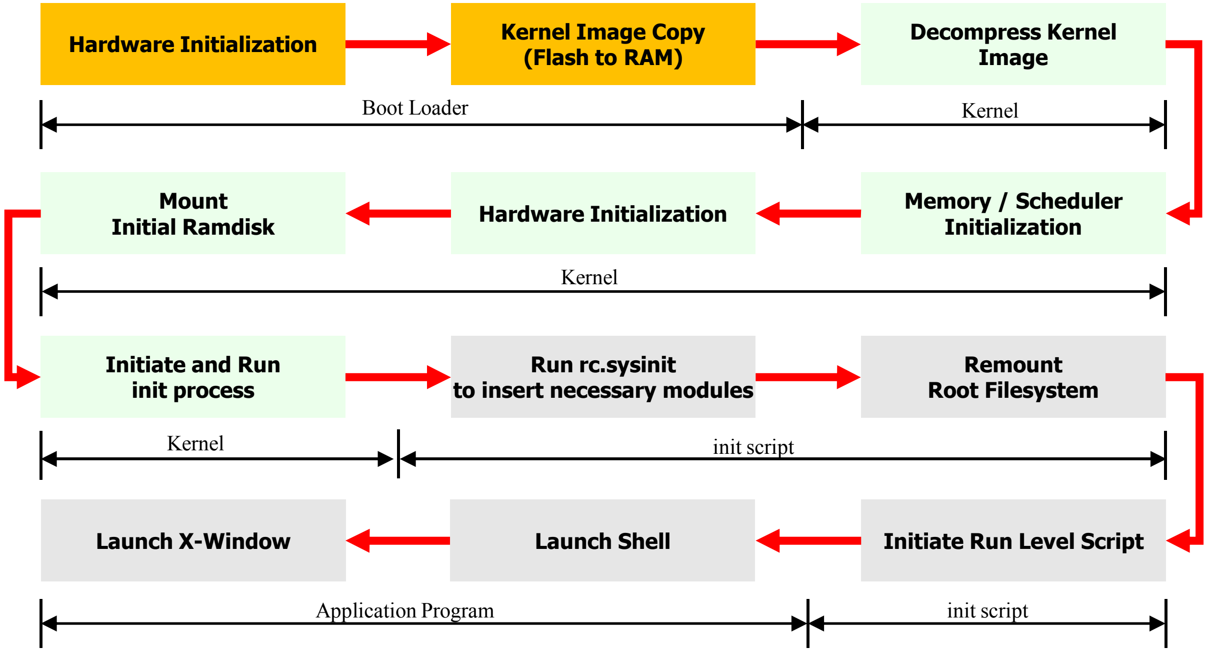
- ❑ Result
- ❑ An analysis
- ❑ Fast booting technology from the viewpoint of mass production
  - ● novice's fault
  - ● mass production?
  - ● Apply with fast booting on mass production point

Fast booting? (1/2)

Booting

- A bootstrapping process that start operating systems when the user turns on a computer system(http://en.wikipedia.org/wiki/Booting)
- Bootstrapping : refers to the start up process a computer uses to load the operating instructions

A boot sequence in the normal manner of linux



Hardware Initialization → Kernel Image Copy (Flash to RAM) → Decompress Kernel Image

Boot Loader | Kernel

Memory / Scheduler Initialization → Hardware Initialization → Mount Initial Ramdisk

Kernel

Initiate and Run init process → Run rc.sysinit to insert necessary modules → Remount Root Filesystem

Kernel | init script

Initiate Run Level Script → Launch Shell → Launch X-Window

Application Program | init script

Fast booting? (2/2)

- ❑ Fast booting
  - Objective
    - Reach from power-on to user functional capability using simplification and optimization of booting sequence as soon as possible
  - Terms
    - User should not sense environments between applied fast booting and non-applied
    - No modification of hardware and no supplements to apply fast booting
- ❑ Differences between embedded systems and PCs (from the technology application aspect)
  - Less computing performance (vs PC)
  - A NAND flash memory file system
  - apply with fast booting technology on a embedded systems with less computing performance
  - Minimization of condition changes along the final state of booting
  - Should be ready to take a sacrifice flexibility

An introduction of fast booting (1/2)

Fast booting technologies on PCs

- Features
  - Common uses like servers, desk-top PCs
  - Various version of kernel, A lot of service, various behavior of application program
- Limitations
  - Users should control boot sequences, not developers
  - There's no sense that fast booting technologies to harm flexibility
- Objectives
  - Should apply fast booting technologies with no harm flexibility
  - User should control boot sequences
    - Script from of boot sequence, not program languages
- technologies
  - initng (http://www.initng.org)
  - Suspend-to-Disk (TuxOnIce)(http://www.suspend2.net)
  - Suspend-to-RAM
  - Ram Disk (initrd)
  - Simplification and optimization of boot sequence scripts

An introduction of fast booting (2/2)

❑ Fast booting technologies on embedded systems

- Features
  - Special Purpose like Mobile Device, Factory Automation, etc…
  - Specified kernel, services, file systems assigned vendor
- Limitations
  - User should no control of boot sequence
  - On developing has finished, boot sequence has been fixed.
  - Error of boot sequence is very critical
- Objectives
  - Technologies applied on the point of finishing developing.
    - No more changes and modifications on boot loader, kernel, rootfs, etc…
  - User should not control boot sequences
    - We can use boot sequence based on non-script.
  - On the debugging point of view, boot sequence can be recreated by tool.
- technologies
  - XIP(http://www.ucdot.org/article.pl?sid=02/08/28/0434210&mode=thread)
  - cramfs (Read Only File System)
  - http://www.celinuxforum.org/CelfPubWiki/BootupTimeResources

Useful fast booting technologies on embedded systems (1/12)

Summary

- Boot loader
  - Removing waiting time
  - Removing unnecessary initialization routines
  - Non-compressed kernel image loading
  - Using optimized copy routine (DMA, Polling)
- kernel
  - Removing unnecessary function and device drivers
  - Modularization of device drivers
  - Avoiding performance measurement routine (BogoMIPS)
  - Removing unnecessary message printout
- File system
  - Using read-only file system
  - Using lazy mount technique on R/W file systems
- Initial script
  - Using binary script, not shell script
  - Using init process with simplified and optimized
- Shared libraries
  - Using prelinking
  - Using preloading or readahead
- Optimization of application programs

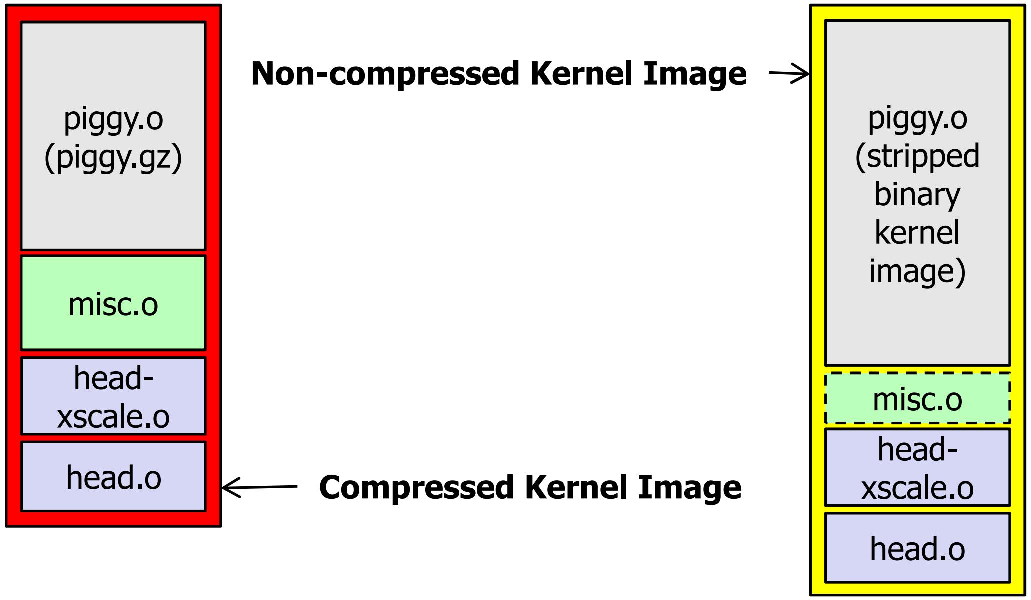
Useful fast booting technologies on embedded systems (2/12)

❑ Boot loader

- Summary
  - A boot loader places kernel into memory.
  - We should remove various level of initialization
- Removing waiting time
  - Description
    - Boot loader await a couple of seconds to connect debug port
    - Since boot loader cannot detect connection of debug port, boot loader awaits a couple of seconds on normal boot sequence.
  - Method examples
    - U-boot : setenv bootdelay 0
    - BLOB : Modify source code directly
- Removing unnecessary initialization routines
  - Description
    - Should not perform initialize other devices except for necessaries to load kernel
  - Method examples
    - Removing initialization of LCD
    - Removing initialization of timer

Useful fast booting technologies on embedded systems (3/12)

- Non-compressed kernel image loading
  - Description
    - Build kernel image without compress option
    - It should be fast boot, but image size has been larger.
    - It should be different results, performance of CPUs, speed of flash memory
  - Method examples
    - Use ‘make Image’ instead of ‘make zImage’



Useful fast booting technologies on embedded systems (4/12)

- Using optimized copy routine
  - Description
    - Direct addressing on NOR Flash Memory like DRAM
    - but, cannot access direct access NAND Flash Memory
    - Sequence in a normal manner
      - » Initialization of Flash → Setup address of flash on flash address register → RnB Set and wait → Copy data register into memory
  - Method examples
    - Copying one time of bunch of memories
    - Using DMA routine
      - » Initialization of DMA channel → setup DMA channel to specify copy range → Enable DMA channel → wait until finishing copy →interrupt
      - » There's a lot of question marks on this technologies

Useful fast booting technologies on embedded systems (5/12)

Kernel

- Summary
  - Remove out other functions which will not be used without necessary.
- Removing unnecessary function and device drivers
  - Description
    - Routines which is not used is waste → Decrease speed of loading
    - better small size of kernel image
  - Method examples
    - Removing unusable Kconfig variables in menuconfig of kernel
    - It should be much of learning by trial and errors
- Modularization of device drivers
  - Description
    - Upload module after boot sequence finished
    - ex : Sound device, CAM device, etc… (unnecessary for booting)
  - Method examples
    - Mark module functions in menuconfig of kernel
    - make INSTALL_MOD_PATH=root_of_rootfs modules_install

Useful fast booting technologies on embedded systems (6/12)

- Avoiding performance measurement routine (BogoMIPS)
  - Description
    - kernel updates loops_per_jiffy value at every booting to using timer (BogoMIPS)
    - This value should be fixed until hardware changes or modify setting
    - The problem is this routine uses some loops (Delaying)
  - Method examples
    - Find out loops_per_jiffy after basic boot sequence
    - Modify init/calibrate.c in kernel source
    - Attach lpj=number on kernel boot parameter → bypass effect
    - ex : lpj=1327104 on 533MHz@SMDK6400
- Removing unnecessary message printout
  - Description
    - a small delaying time with printout through the serial port
    - 0.5sec delay at 115200bps (printout about 7000 or more characters)
  - Method examples
    - Attach "quiet" on kernel boot parameter (It cannot printout on level 4 messages)
    - If you want to every message to avoid →loglevel=0

Useful fast booting technologies on embedded systems (7/12)

❑ File System

- Summary
  - a mount is necessary to use file system
  - in R/W file system, there are much of booting time to mount that, and avoid
- Using Read-only File System
  - Description
    - Without using write functionality, simplify using file system
    - There are no delay time on mount read-only file systems
    - One of simplest, RomFS
    - The cramfs has compression functionality
    - Avoid ram disk file system, copying flash into memory at boot time
  - Method examples
    - Classify read-only, R/W, temporary files that will include file system
    - Position cramfs read-only files
    - Using lazy mount technique, to include R/W files
    - Temporary file uses tmpfs
    - At boot sequence, avoid copy cramfs into tmpfs (delay time)

Useful fast booting technologies on embedded systems (8/12)

- Using lazy mount technique on R/W file systems
  - Description
    - In some cases, there must be write and keep files
    - ex : /etc/pointercal (tslib)
    - Write through lazy mount technique
    - Lazy Mount?
      - slow mount speed file systems like jffs2 mount after finishing boot sequence on user request
      - On booting, using read-only file instead
  - Method examples
    - Assume pointercal(cramfs) file is in /etc/touch
    - at booting : there are read-only file in /etc/touch
    - at finishing booting : mount –t jffs2 /dev/mtdblock4 /etc/touch → Reuse pointercal(jffs2)

Useful fast booting technologies on embedded systems (9/12)

❑ Initial Script

- Summary
  - Simplify and Optimize initial script
- Using binary script
  - Description
    - a flexible shell script uses interpreter
    - init process performs system initialization through /etc/inittab, and perform rc.sysinit
    - at this point, init process must include shell(to perform shell script)
  - Disadvantages performance of shell script
    - Interpreter
      - » Interpret every line and perform
      - » Interpret meaningless line and phrases
    - Use fork & exec technique when perform commands
      - » It should be heavy load when fork in embedded system which has big load using TLB
    - Heavy Shell
      - » To perform shell script, a shell has loader, interpreter, singaling, etc..
  - Method examples
    - Make binary script without fork & exec technique

Useful fast booting technologies on embedded systems (10/12)

- Using init process with simplified and optimized
  - Description
    - Busybox, a popular shell and commands in embedded systems
    - Busybox includes init process
    - Also busybox includes functionality of shell, at boot sequence, it is big monster of delaying boot time
    - ex : a size of busybox is 1.5MB(static) at SMDK6400
  - Method examples
    - Combine between init process and binary script, and replace /sbin/init
    - Main functionalities of init process
      - » Perform initial script
      - » Perform signaling when child process has been killed
      - » Perform respawn every 1sec after watching
      - » Perform system halt & reboot when self destruction

Useful fast booting technologies on embedded systems (11/12)

❑ Shared Libraries

- Summary
  - Most of application programs include minimized code, data, and bit portion of shared libraries.
  - /proc/process_number/maps
  - Shared library are big monster when perform boot sequence, also reading file systems
  - the problem is loading shared library into memory as soon as possible
- Prelinking
  - Description
    - Shared libraries are basically PIC(Position independent code) format
    - When compile & link, prelinking addresses of shared library
    - but, there's no effectable

Useful fast booting technologies on embedded systems (12/12)

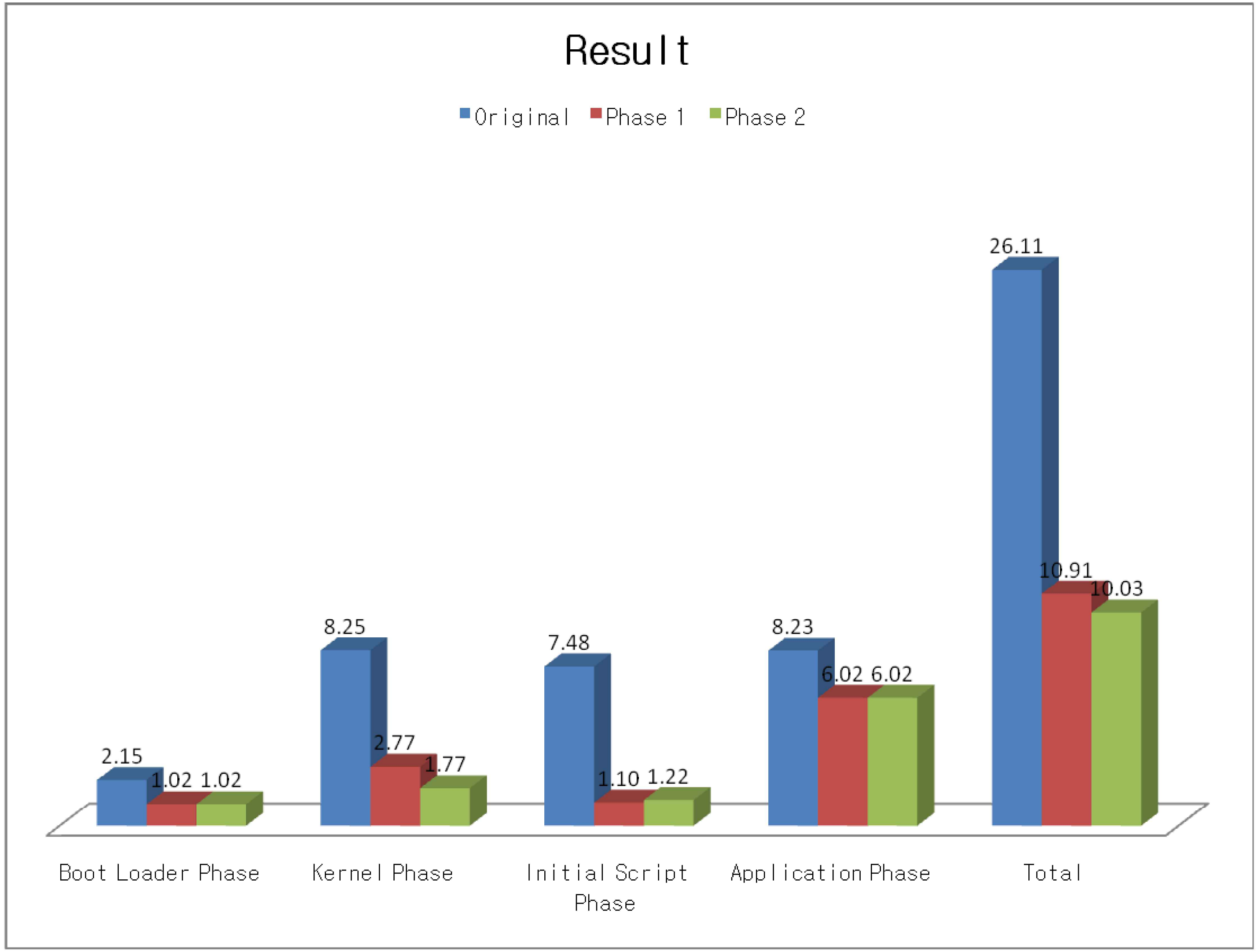
- Preloading or Readahead
  - Description
    - Basically, shared library loaded into memory with mmap method
    - Before starting process, load into memory shared libaries and can be fast launching processes
  - Problem
    - At before and after the launching process, it should be loaded into flash to memory → there's no performance improvement
  - A reason for this behavior
    - If application program has multi process
      - There must be synchronized when perform sequence of each processes
      - If there cannot be no synchronization, using sleep code
      - It can be reduce time of sleep, when loading shared libraries early
      - ex : Using X-Window and Matchbox Window Manager
    - It can be using optimized load method
      - In some cases, DMA transfer is helpful increasing booting speed
      - When CPU is idle, DMA transfer is faster
      - DMA transfer will not be useful, launching application level which has started multitasking
      - Therefore, before launching application program, it can be fast loading through DMA transfer
      - In some case, DMA transfer has 4 times ability than general loading (2MB/sec vs 8MB/sec)
      - When multitasking has started, CPU must working memory, DMA transfer would be worthless

Result (1/2)

	Original	Phase 1	Phase 2
Boot Loader Phase	2.15	1.02	1.02
Kernel Phase	8.25	2.77	1.77
Initial Script Phase	7.48	1.10	1.22
Application Phase	8.23	6.02	6.02
Total	26.11	10.91	10.03

Original : Initially released version
Phase 1 : initial script applied
Phase 2 : initial script applied and simplified init process

Result (2/2)


Result
🟦Original 🟥Phase 1 🟩Phase 2
2.15
1.02
1.02
8.25
2.77
1.77
7.48
1.10
1.22
8.23
6.02
6.02
26.11
10.91
10.03
Boot Loader Phase
Kernel Phase
Initial Script Phase
Application Phase
Total

An analysis

An analysis

- Boot Loader Phase
  - $(2.15_{\text{original}} \rightarrow 1.02)$ sec
  - There are still 1 sec or more “boot delay is 0”
  - Suppose that kernel image copy
    - # nand read 0x40000 0x1c0000 0xc0008000 ; bootm 0xc0…
- Kernel Phase
  - $(8.25_{\text{original}} \rightarrow 2.77_{\text{1st}} \rightarrow 1.77_{\text{2nd}})$ sec
  - More effectiveness jffs2 → cramfs (Phase 1)
  - Small effectiveness kernel modularization and removal of unused functionality (Phase 2)
- Initial Script Phase
  - $(7.48_{\text{original}} \rightarrow 1.10_{\text{1st}} \rightarrow 1.22_{\text{2nd}})$ sec
  - More effectiveness binary script(Phase 1)
  - It takes more small time : effect of module upload
- Application Phase
  - $(8.23_{\text{original}} \rightarrow 6.02_{\text{1st}} \rightarrow 6.02_{\text{2nd}})$ sec
  - Effect of binary script

Fast booting technology from the viewpoint of mass production(1/2)

❑ Novice's fault

- "That's are… everyone can do it"
  - Structural complexity, not difficulty
  - Technique applied level is a lot composition and cross dependancy
- "I am so annoying, bypass that"
  - Thinking developer's place instead customer's place
  - a lack of catching importance of fast booting
- "Where is him? He applied fast booting on our system last time."
  - A special case of IT industry. The turnover
  - a lack of formal training
- "I'm not developer… Why he commands this job for me?"
  - Fast booting technology is side of mass production, not developing
  - The engineer of mass production thinks that developing is not range of work

❑ What is mass production?

- a specialized stage of production, phase of production of product
- more specialized and divide, big industry

Fast booting technology from the viewpoint of mass production(2/2)

❑ Apply with fast booting on mass production point

- To solve that structural complexity with technology application
  - A specialized tool of fast booting and documentation
- To solve that cognition of importance of fast booting
  - Train applying fast booting on general training course and reeducation program in company
  - Improve fast booting technology through out rapidly developing
- Applying specialized fast booting
  - Needs documentation and form of formal job, not person to person
  - Rapidly update fast booting tool, to apply mass production phase using developing phase
- Easy to use
  - Develop easy tool that can use mass production engineer
  - Can be estimated boot time
  - Develop various fast booting technique to satisfy customer's requests

Thank you!

Q&A