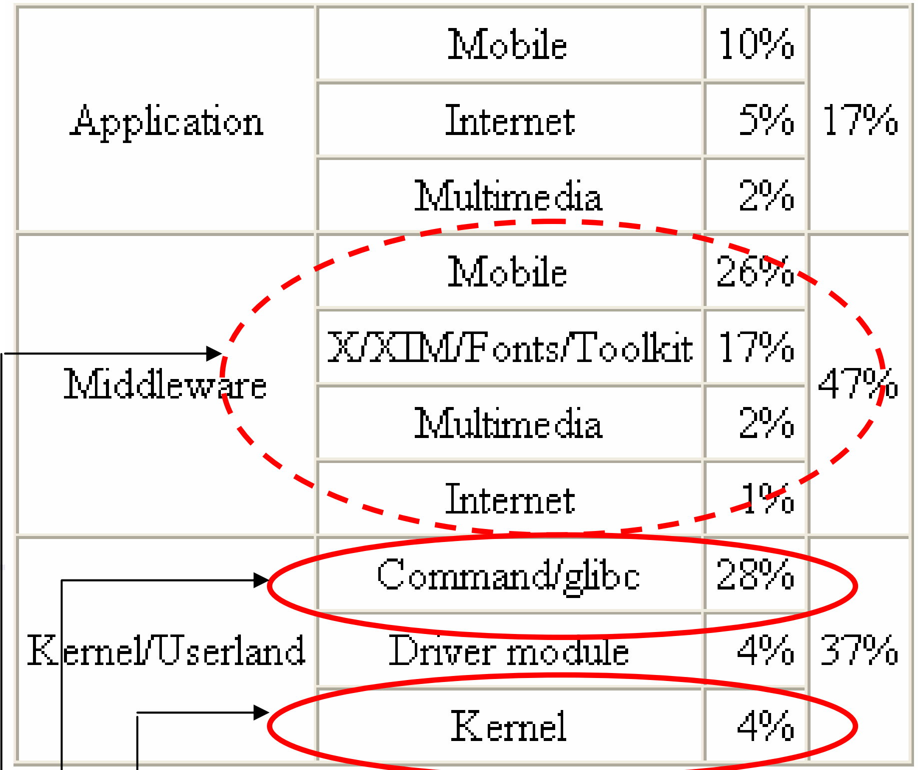
2005.01.26

# SZWG Meeting at Plenary Meeting

Hiro Suyama

# Observation from Profiling

Refresh Memory



| | | | |
|---|---|---|---|
| Application | Mobile | 10% | 17% |
| | Internet | 5% | |
| | Multimedia | 2% | |
| Middleware | Mobile | 26% | 47% |
| | X/XIM/Fonts/Toolkit | 17% | |
| | Multimedia | 2% | |
| | Internet | 1% | |
| Kernel/Userland | Command/glibc | 28% | 37% |
| | Driver module | 4% | |
| | Kernel | 4% | |

Kernel size is relatively small issue from this table, however, looking at the absolute number of 1M+ byte is still problem.

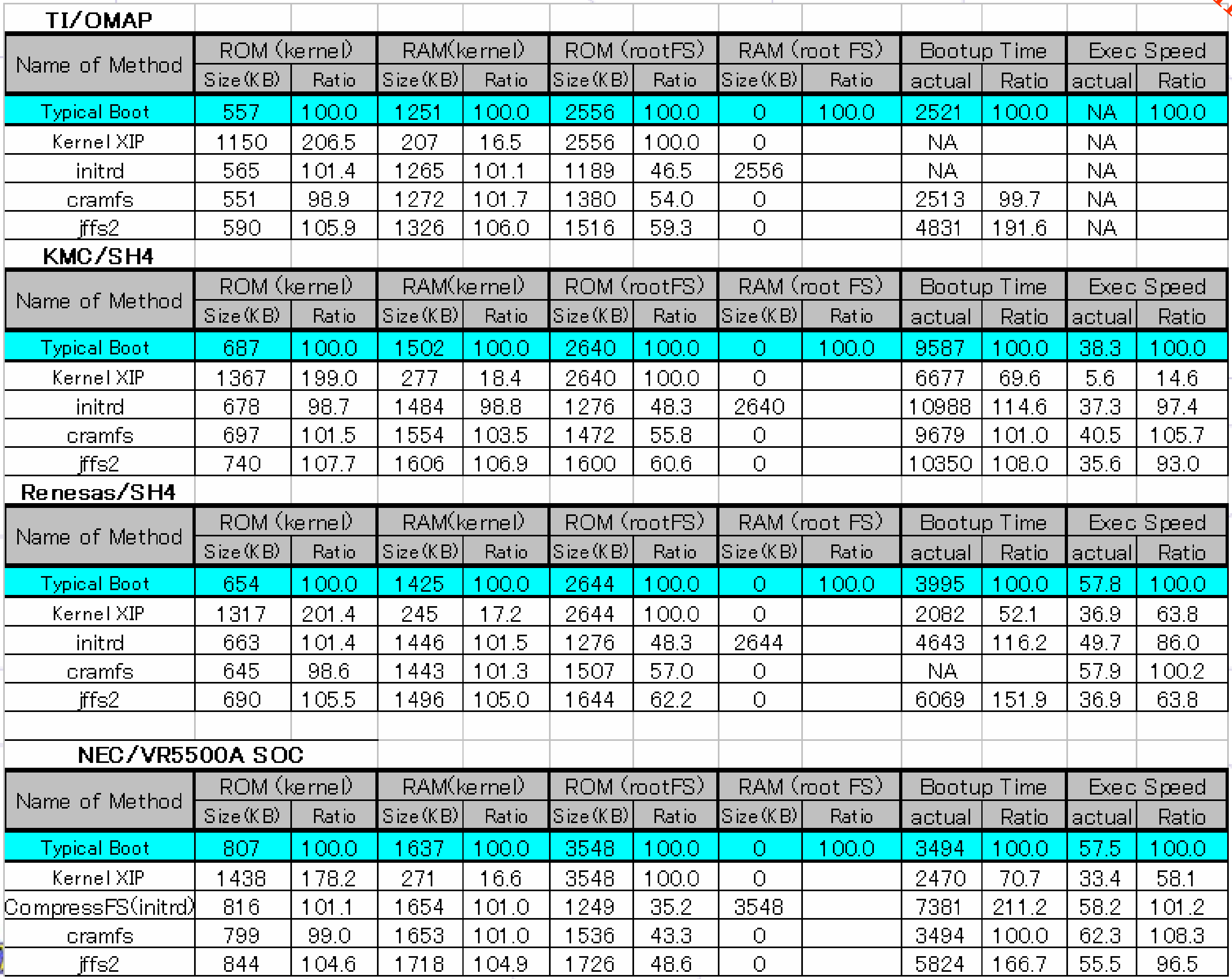
Rootfs looks bigger fat

Middleware is another big fat ( outside the scope of SZWG)

# Result Summary

## zImage vs Kernel XIP, initrd vs cramfs vs jffs2

Refresh Memory



**TI/OMAP**

| Name of Method | ROM (kernel) | | RAM(kernel) | | ROM (rootFS) | | RAM (root FS) | | Bootup Time | | Exec Speed | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Size(KB) | Ratio | Size(KB) | Ratio | Size(KB) | Ratio | Size(KB) | Ratio | actual | Ratio | actual | Ratio |
| Typical Boot | 557 | 100.0 | 1251 | 100.0 | 2556 | 100.0 | 0 | 100.0 | 2521 | 100.0 | NA | 100.0 |
| Kernel XIP | 1150 | 206.5 | 207 | 16.5 | 2556 | 100.0 | 0 | | NA | | NA | |
| initrd | 565 | 101.4 | 1265 | 101.1 | 1189 | 46.5 | 2556 | | NA | | NA | |
| cramfs | 551 | 98.9 | 1272 | 101.7 | 1380 | 54.0 | 0 | | 2513 | 99.7 | NA | |
| jffs2 | 590 | 105.9 | 1326 | 106.0 | 1516 | 59.3 | 0 | | 4831 | 191.6 | NA | |

**KMC/SH4**

| Name of Method | ROM (kernel) | | RAM(kernel) | | ROM (rootFS) | | RAM (root FS) | | Bootup Time | | Exec Speed | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Size(KB) | Ratio | Size(KB) | Ratio | Size(KB) | Ratio | Size(KB) | Ratio | actual | Ratio | actual | Ratio |
| Typical Boot | 687 | 100.0 | 1502 | 100.0 | 2640 | 100.0 | 0 | 100.0 | 9587 | 100.0 | 38.3 | 100.0 |
| Kernel XIP | 1367 | 199.0 | 277 | 18.4 | 2640 | 100.0 | 0 | | 6677 | 69.6 | 5.6 | 14.6 |
| initrd | 678 | 98.7 | 1484 | 98.8 | 1276 | 48.3 | 2640 | | 10988 | 114.6 | 37.3 | 97.4 |
| cramfs | 697 | 101.5 | 1554 | 103.5 | 1472 | 55.8 | 0 | | 9679 | 101.0 | 40.5 | 105.7 |
| jffs2 | 740 | 107.7 | 1606 | 106.9 | 1600 | 60.6 | 0 | | 10350 | 108.0 | 35.6 | 93.0 |

**Renesas/SH4**

| Name of Method | ROM (kernel) | | RAM(kernel) | | ROM (rootFS) | | RAM (root FS) | | Bootup Time | | Exec Speed | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Size(KB) | Ratio | Size(KB) | Ratio | Size(KB) | Ratio | Size(KB) | Ratio | actual | Ratio | actual | Ratio |
| Typical Boot | 654 | 100.0 | 1425 | 100.0 | 2644 | 100.0 | 0 | 100.0 | 3995 | 100.0 | 57.8 | 100.0 |
| Kernel XIP | 1317 | 201.4 | 245 | 17.2 | 2644 | 100.0 | 0 | | 2082 | 52.1 | 36.9 | 63.8 |
| initrd | 663 | 101.4 | 1446 | 101.5 | 1276 | 48.3 | 2644 | | 4643 | 116.2 | 49.7 | 86.0 |
| cramfs | 645 | 98.6 | 1443 | 101.3 | 1507 | 57.0 | 0 | | NA | | 57.9 | 100.2 |
| jffs2 | 690 | 105.5 | 1496 | 105.0 | 1644 | 62.2 | 0 | | 6069 | 151.9 | 36.9 | 63.8 |

**NEC/VR5500A SOC**

| Name of Method | ROM (kernel) | | RAM(kernel) | | ROM (rootFS) | | RAM (root FS) | | Bootup Time | | Exec Speed | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Size(KB) | Ratio | Size(KB) | Ratio | Size(KB) | Ratio | Size(KB) | Ratio | actual | Ratio | actual | Ratio |
| Typical Boot | 807 | 100.0 | 1637 | 100.0 | 3548 | 100.0 | 0 | 100.0 | 3494 | 100.0 | 57.5 | 100.0 |
| Kernel XIP | 1438 | 178.2 | 271 | 16.6 | 3548 | 100.0 | 0 | | 2470 | 70.7 | 33.4 | 58.1 |
| CompressFS(initrd) | 816 | 101.1 | 1654 | 101.0 | 1249 | 35.2 | 3548 | | 7381 | 211.2 | 58.2 | 101.2 |
| cramfs | 799 | 99.0 | 1653 | 101.0 | 1536 | 43.3 | 0 | | 3494 | 100.0 | 62.3 | 108.3 |
| jffs2 | 844 | 104.6 | 1718 | 104.9 | 1726 | 48.6 | 0 | | 5824 | 166.7 | 55.5 | 96.5 |

# Activity past 6 months

- Data Measurement on 2.6 Kernel
  - Measure 2.6 Kernel data to compare the trend
  - (zImage, Kernel XIP,initrd, cramfs, jffs2) x
    - ( K-ROM, K-RAM, Rtfs-ROM, Rtfs-RAM,boot, exec speed)
- Linux Tiny
  - Prioritize patches for embedded platform
    - Static RAM/ROM reduction
    - Dynamic RAM reduction
  - Port those patches to several embedded platform
    - Renesas SH4 board
    - Toshiba TX49(MIPS) board
- Application XIP
  - cramfs with linear option patch released
- Squashfs
  - Partial data captured
- glibc
  - Experience shared on optimization, glibc vs uClibc

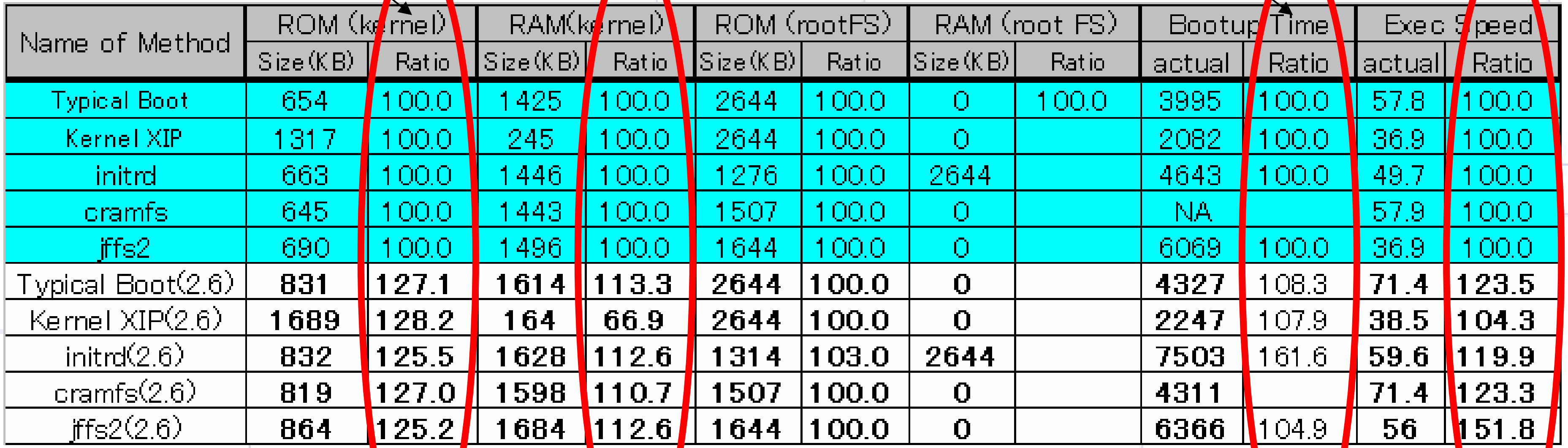
# Data Measurement on 2.6 Kernel

## - 2.4 vs 2.6 -

Kernel ROM size become about 25% bigger

Bootup time become worse

Renesas SH4 Platform



| Name of Method | ROM (kernel) | | RAM(kernel) | | ROM (rootFS) | | RAM (root FS) | | Bootup Time | | Exec Speed | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Size(KB) | Ratio | Size(KB) | Ratio | Size(KB) | Ratio | Size(KB) | Ratio | actual | Ratio | actual | Ratio |
| Typical Boot | 654 | 100.0 | 1425 | 100.0 | 2644 | 100.0 | 0 | 100.0 | 3995 | 100.0 | 57.8 | 100.0 |
| Kernel XIP | 1317 | 100.0 | 245 | 100.0 | 2644 | 100.0 | 0 | | 2082 | 100.0 | 36.9 | 100.0 |
| initrd | 663 | 100.0 | 1446 | 100.0 | 1276 | 100.0 | 2644 | | 4643 | 100.0 | 49.7 | 100.0 |
| cramfs | 645 | 100.0 | 1443 | 100.0 | 1507 | 100.0 | 0 | | NA | | 57.9 | 100.0 |
| jffs2 | 690 | 100.0 | 1496 | 100.0 | 1644 | 100.0 | 0 | | 6069 | 100.0 | 36.9 | 100.0 |
| Typical Boot(2.6) | 831 | 127.1 | 1614 | 113.3 | 2644 | 100.0 | 0 | | 4327 | 108.3 | 71.4 | 123.5 |
| Kernel XIP(2.6) | 1689 | 128.2 | 164 | 66.9 | 2644 | 100.0 | 0 | | 2247 | 107.9 | 38.5 | 104.3 |
| initrd(2.6) | 832 | 125.5 | 1628 | 112.6 | 1314 | 103.0 | 2644 | | 7503 | 161.6 | 59.6 | 119.9 |
| cramfs(2.6) | 819 | 127.0 | 1598 | 110.7 | 1507 | 100.0 | 0 | | 4311 | | 71.4 | 123.3 |
| jffs2(2.6) | 864 | 125.2 | 1684 | 112.6 | 1644 | 100.0 | 0 | | 6366 | 104.9 | 56 | 151.8 |

| | |
|---|---|
| 2.4 | 2.4.20 |
| 2.6 | 2.6.7 |

Kernel RAM size become about 10% bigger

Execution Speed improved

# Linux Tiny Scope

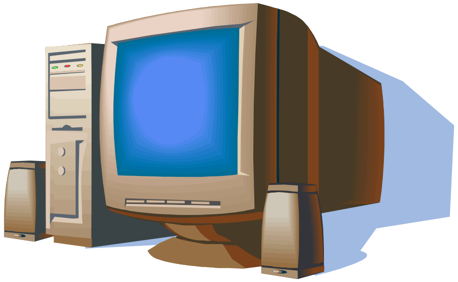
## Trial #1 Reduce Static RAM/ROM

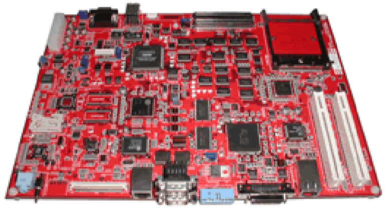
### Step1

Develop the "script" which automate the process of 1) apply and build linux tiny patch individually 2) Capture static RAM/ROM size and create record

### Step2

Create individual config . Actually apply, build and capture the data. Determine "top n" patches





### Step3

Port "top n" patches onto embedded platform and measure the size effect. Also measure side effect such as bootup time, execution speed.

## Trial #2 Reduce Dynamic RAM

Try replace memory allocator from SLAB to SLOB on the embedded platform.

# Linux Tiny Status

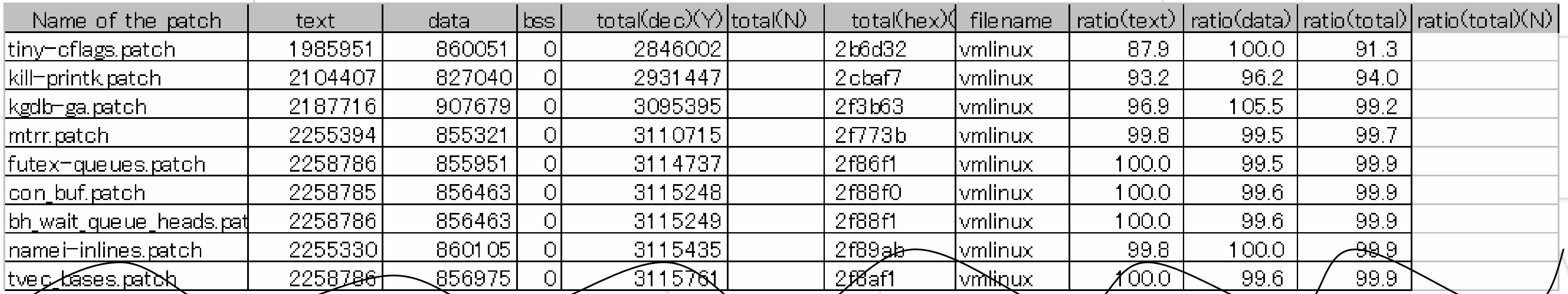
Step 1 and 2 completed !!

## Executive Summary

2.6.10-tiny1.patch

| Patch Name | Effect | | Note |
|---|---|---|---|
| tiny-cflags | 273K | 9.7% | x86 depend |
| kill-printk | 187K | 6.0% | |
| Removal ex-POSIX and POSIX timer group patch | 53K | 1.7% | |
| No-bug group patch | 37K | 1.2% | |
| Tiny-VT | 33K | 1.05% | |

## Raw Data



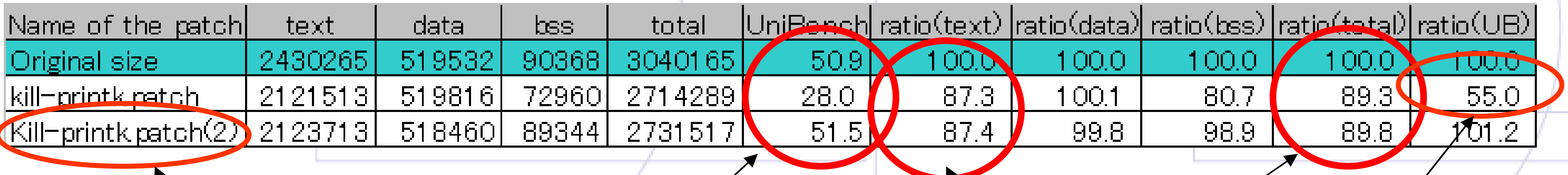
| Name of the patch | text | data | bss | total(dec)(Y) | total(N) | total(hex)( | filename | ratio(text) | ratio(data) | ratio(total) | ratio(total)(N) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| tiny-cflags.patch | 1985951 | 860051 | 0 | 2846002 | | 2b6d32 | vmlinux | 87.9 | 100.0 | 91.3 | |
| kill-printk.patch | 2104407 | 827040 | 0 | 2931447 | | 2cbaf7 | vmlinux | 93.2 | 96.2 | 94.0 | |
| kgdb-ga.patch | 2187716 | 907679 | 0 | 3095395 | | 2f3b63 | vmlinux | 96.9 | 105.5 | 99.2 | |
| mtrr.patch | 2255394 | 855321 | 0 | 3110715 | | 2f773b | vmlinux | 99.8 | 99.5 | 99.7 | |
| futex-queues.patch | 2258786 | 855951 | 0 | 3114737 | | 2f86f1 | vmlinux | 100.0 | 99.5 | 99.9 | |
| con_buf.patch | 2258785 | 856463 | 0 | 3115248 | | 2f88f0 | vmlinux | 100.0 | 99.6 | 99.9 | |
| bh_wait_queue_heads.pat | 2258786 | 856463 | 0 | 3115249 | | 2f88f1 | vmlinux | 100.0 | 99.6 | 99.9 | |
| namei-inlines.patch | 2255330 | 860105 | 0 | 3115435 | | 2f89ab | vmlinux | 99.8 | 100.0 | 99.9 | |
| tvec_bases.patch | 2258786 | 856975 | 0 | 3115761 | | 2f8af1 | vmlinux | 100.0 | 99.6 | 99.9 | |

# Linux Tiny Status

Step 3 partially been tried !!

Potential room for contribution been found.

data on embedded platform (SH4)



| Name of the patch | text | data | bss | total | UniBench | ratio(text) | ratio(data) | ratio(bss) | ratio(total) | ratio(UB) |
|---|---|---|---|---|---|---|---|---|---|---|
| Original size | 2430265 | 519532 | 90368 | 3040165 | 50.9 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| kill-printk patch | 2121513 | 519816 | 72960 | 2714289 | 28.0 | 87.3 | 100.1 | 80.7 | 89.3 | 55.0 |
| Kill-printk patch(2) | 2123713 | 518460 | 89344 | 2731517 | 51.5 | 87.4 | 99.8 | 98.9 | 89.8 | 101.2 |

Modification done for console operation (by Mitsubishi) and improved the performance

Good Result !!

**Oops !!**

Note:
bigger number = higher performance

## Trial #1 Save Static ROM/RAM usage

### Step 3

Which patches to port ?

- kill-printk
- POSIX related group patch
- No-bug group patch
- Tiny-VT

High Priority Candidate

Which Platform ?

Who to port ?

## Trial #2 Save Dynamic RAM allocation

Which patches to port ?

- Replace SLAB to SLOB

How Can we measure the result ?

- Which tool to be used ?
- Which application to be used ?

Which Platform ?

Who to port ?

# GLIBC

## (1) Replace GLIBC with uClibC

Could achieve 897K size reduction

Had Problem with 3rd parties binary

Motorola’s soluiton

1) Rebuild each component – Request 3rd Party vendors to rebuild
2) Modify uClibc to be API compatible with glibc ,including adding a versioning system and structure modification.
3) Write a light weight “translation” or “pass through version of glibc that satisfies the requirements of each executable are met, but that calls the uClibc library to perform the necessary work.

1) is not feasible solution as we may not be able to get 3rd party to agree to build all the binaries and resolve issues. Its an expensive solution.
2) We have limited resource/time to put our efforts in adding api's And making uClibc compatible.
3) this is what we have been playing with.. we set a goal of building some user apps with a lightweight version of glibc and tried to port some ulibc functionality. Again we do not have resource/time at this time to test them thoroughly and make it more generic... its more of a hack right now.

# Ideas on small-library compatibility with glibc, from an expert

http://tree.celinuxforum.org/CelfPubWiki/SubsetLibcSpecification

> Possible Solution:
> 1) Rebuild each component.Request 3rdpart vendors to rebuild.
> 2) Modify uClibcto be API compatible with glibc, including adding a versioning system and structure
> modification.
> 3) Write a lightweight "translation" or "pass-through" version of glibc that satisfies the requirements of each executable
> are met, but that calls the uClibclibrary to perform the necessary work.

I strongly recommend #1. Recompiling applications with uClibc is almost always very easy to do for applications
that already compile with glibc.
If the vendor is not technically capable of doing the needed work, I have a consulting company that would be happy to
Provide assistance to 3rd vendors and to Motorola. :-)

> As I understood, uClibc - from API point of view - is very
> close to glibc. Which part can be incompatible ?

uClibc and glibc have nearly identical APIs. With a very few
exceptions, almost any program that will compile with glibc
will also compile with uClibc.

http://www.uclibc.org/cgi-bin/cvsweb/uClibc/docs/Glibc_vs_uClibc_Differences.txt?view=auto

# Ideas on small-library compatibility with glibc, from an expert

http://tree.celinuxforum.org/CelfPubWiki/SubsetLibcSpecification

> Is there a way to make uClibc fully compatible with glibc ?

In my opinion, uClibc _is_ compatible with glibc. But it is compatible at the source code (API) level. Most code can be easily recompiled vs the latest uClibc. What you are really asking about is binary, or ABI compatibility.
The largest issues preventing uClibc from having an ABI that is 100% binary compatible with glibc are the following things.

1) Naming. uClibc's shared library loader, C library, and even start up functions are named differently from their glibc counterparts.
2) uClibc sometimes uses different opaque data types than glibc.
3) uClibc directly uses the linux kernel's arch specific data structures, such as 'stuct stat', while glibc almost always translates kernel data structures into separate user space data structures. This causes uClibc to be somewhat more tightly coupled with a particular kernel major version (2.2.x, 2.4.x, 2.6.x) than glibc. When changing from 2.4.x to 2.6.x, it is advisable to recompile uClibc.
4) uClibc's stdio code is completely different from glibc's. This causes significant ABI differences for functions that are possible pthread cancellation points, for functions that are allowed to be macros by SuSv3. Additionally, uClibc allows BUFSIZ to be set to values different from that used by glibc. Some stdio functions, such as fcloseall() and __fpending() can behave differently than their glibc counterparts. Other stdio functions, such as fscanf() behave differently in cases where glibc does not comply with SuSv3.
5) /etc/timezone and the whole zoneinfo directory tree are not supported by uClibc. uClibc uses /etc/TZ, set per the value of the TZ env variable, per SuSv3.
6) Symbol versioning. All glibc symbols have specific symbol versioning applied, so glibc does not have an 'fopen' symbol, but rather has a 'fopen@GLIBC_2.0' symbol. In some cases, such as with 'sys_siglist', glibc has a number of implementations of the same symbol (sys_siglist@GLIBC_2.0, sys_siglist@GLIBC_2.1, and sys_siglist@@GLIBC_2.3.3) in order to maintain ABI compatibility with earlier versions of glibc.

doubtless there are other reasons why uClibc's ABI does not and will not easily match the glibc ABI.

Can We, as SZWG recommend uClibc as preferable Solution for CE devices and encourage 3rd party vender to switch to uClibc ?

# GLIBC

## (2) Optimize glibc(or uClibc)

Could achieve100K size reduction

Had the following problem

- Requires rerun of the tool on each version of software release.
    - Dynamic loading off apps may be a problem.

This solution would be okay for closed system.
Would be good idea to define optimized lib based on product profile.
SZWG may collabrate with MPPWG to define optimized lib for mobile phone.

# Application XIP

Status The patch of cramfs with linear option is available.
Nice to have measurement on size and side effect.

# Squashfs

Status Some experience shared. The number look attractive.
Nice to have measurement on size and side effect.

| | |
|---|---|
| Motorola’s case | 2.55M(cramfs) → 2.27M(Squashfs) 11% |
| Sony’s case | 57% reduction compared with ext2(?) |