Looking into Linux Tiny

July 15, 2005

Linux Promotion Center, NEC

Munehiro Ikeda

m-ikeda(@)ds.jp.nec.com

(Volunteer in translation from Japanese to English : S. Ueda, Sony)

0. Abstract of the Session

LinuxTiny is the collection of the patches to reduce the system size and memory consumption of the Kernel.

In this session, we will introduce the result of the research on the contents of the LinuxTiny performed by us, Linux Technology Center of NEC.

- Necessity of Memory size reduction
- Outline of LinuxTiny
- Category of the patches by purpose
- Scale of the patches
- Proving the effect of Kernel Size and Memory Size reduction
  - Effective Settings / Patches
  - Other Settings / Patches
- What are planned next?

1. Necessity of Memory size reduction

- Memory size and storage size of ordinary PC are drastically increasing.
- Even Linux which have been believed “light weighted” is also increasing the kernel size as well as the memory size in accordance with the enhancement of the features.
- However the strong demand still exists such as:
  - “I would like to use Linux on my old fashioned PC, even though the Hard disk capacity and memory size are small”
  - “I wish to experience the latest version Linux, 2.6!!”
- For those demands, squeezing the Kernel size and reduction of the memory size are required.

-> Investigation of those reduction by Linux Tiny

This direction is also fitting to the demand from the Embedded system use of Linux

2. Outline of LinuxTiny

- Developed by Matt Mackall
  http://www.selenic.com/tiny-about/
- Collection of patches to ultimately minimize the Kernel size and the memory consumption
  - 120 patches, 369 files retouched
- Abstract of the features
  - For embedded system usage (size reduction, less memory consumption)
    - Enable to set removing printk(), BUG and panic() in order to minimize Kernel size
    - Enable to set removing not usually used features to minimize Kernel size (ptrace, direct I/O, POSIX Timer, IGMP, rtnetlink etc.)
    - Changed SLAB allocator to SLOB allocator which enables better efficient memory use
    - Detected the position of In-Line Function call. Including the tool to calculate size after building up
  - Kexec
    - Enabled to add Kexec for clash dumping
  - Kgdb
    - Enabled to add kgdb. Remote debugging via Ethernet is supported

* Worked on Kernel 2.6.10

3. Category of the patches by purpose

- Categorize the purposes by the setting items upon Kconfig
  - For embedded system
    - General setup --->
      Configure standard kernel features (for small systems) --->
      dozen of setting items added below
    - The objective is to reduce Kernel size and minimizing memory consumption
  - Added kexec function
    - Processor type and features ---> kexec system call
  - Added kgdb function
    - Kernel hacking ---> Include kgdb kernel debugger

4. Scale of the patches

Target of analysis for this study

Category	Number of Patches	Corrected Files	Corrected Lines(+)	Corrected Lines (-)
For Embedded	91	249	8350	2769
kexec	25	71	2179	171
kgdb	3	48	6060	50
Other	1	1	2	2
Total	120	369	16591	2992

Cumulative numbers are shown in “Corrected Files” and “Corrected Lines”.

- Both the maximum number of the patches and corrected items are for embedded system use.
- "Other" is for changing “Makefile EXTRAVERSION” to “-tiny”.

5. Proving the effect of Kernel Size and Memory Size reduction

Compared 3 types of Kernels build

1) Vanilla-full: Reference

- As the typical Linux Kernel
- Making the vanilla Kernel (2.6.10) almost not changing the original setting
  - Loadable_Module_Support = n, Pre_Emptive_Kernel = n, Power_Management = n, Flame_Buffer = n

2) Vanilla-size: To compare with “tiny”

- How “vanilla” can minimize the size?
- The Best setting to minimize the size
  - for small systems = y, Symbol_Load =n, Math_Emulation = n ,
    Optimize for size = y ,
    Minimum required drivers, “fs” is “ext2”, “ext3”, “reiser”, “ISO9660”, “proc”, “sysfs” and “tmpfs” only

3) Tiny: Main target of analysis

- How much the Kernel size minimized applying the LinuxTiny patches?
- All added setting items for applying the vanilla-size setting + LinuxTiny, are set to that we can expect to reduce the size as well as the memory consumption

5.1 Effectiveness of Kernel Size Reduction

- Result of the Kernel Size Measurement
  - Kernel 2.6.10
  - x86(PC compatible) Kernel
  - gcc 3.3.5

	vmlinux[KB]	bzImage[KB]	
(vanilla-full)	(7709)	(3071)	(reference)
vanilla-size	2833	1061	
tiny	2360	869	

=> LinuxTiny enables to build up the Kernel image less than 1MB of version 2.6 Linux.

5.2 Effectiveness of Memory Consumption Reduction

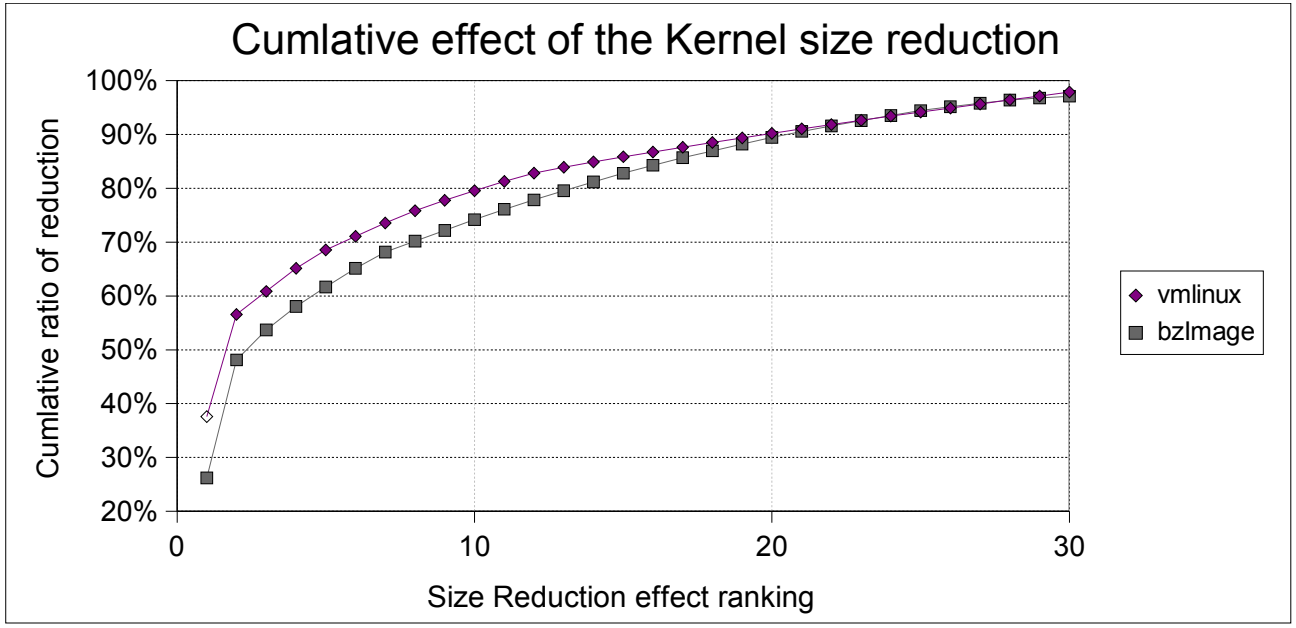
- Result of the measurement of memory consumption
  - Kernel 2.6.10
  - x86(PC compatible) Kernel
  - gcc 3.3.5
  - Machine : 96MB RAM / Celeron 400MHz
  - Stop all daemons (not starting up)
  - Measure in "free" just after Linux started

	total[KB]	used[KB]	free[KB]	buffers[KB]	cached[KB]	
(vanilla-full)	(90200)	(11572)	(78628)	(1324)	(4052)	Reference
vanilla-size	94656	9668	84988	1316	4028	
tiny	95036	8988	86048	1392	4004	

=> More than 1MB free space of memory was brought by LinuxTiny

5.3 Effective settings and patches to minimize Kernel size

- 80% of the kernel size reduction is performed by top 10 settings (patches). (evaluated the effect norming the size difference between vanilla-size and tiny as 100%).
  - Vmlinux : 80% of the effect brought by the top 10 settings
  - bzImage : 74% of the effect brought by the top 10 settings



5.3 Effective settings and patches to minimize Kernel size

1) TINY_CFLAGS = n,TINY_CFLAGS_VAL(tiny-cflags.patch):26.2%
   - Add options to optimize architecture onto CFLAGS (default : "-march=i386"))

2) PRINTK = n (kill-printk.patch) : 21.9%
   - Replace printk(), do_syslog(), sys_syslog() and call_console_drivers() to Empty Functions and remove text string data.
   - Remove printk() call in ineffective portion of the interrupt handler (ignore_int label) in arch/i386/kernel/head.S
   - ✗ Demerit: Trace-ability penalty because dmesg and syslog become ineffective.

3) TINYVT = n (tinyvt.patch) : 5.6%
   - Changed the code of virtual terminal from the standard (vt_ioctl.c, vc_screen.c etc.) to tinyvt.c which the code is very simple.
   - ✗ Demerit: non stable operation. Sometimes it could not boot up.
   - ✗ Demerit: Eliminated from the patches for 2.6.11.

- Evaluated the effect norming the size difference between vanilla-size and tiny as 100%.

5.3 Effective settings and patches to minimize Kernel size

4) RTNETLINK = n (rtnetlink.patch) : 4.4%

- Replaced the function set related rtnetlink such as rtnl(), rtattr_strcmp(), rtnetlink_send() etc. to null macro or in_line function which simply returns 0, __rtnull().
- DIRECTIO = n (direct-io-core.patch) : 3.6%
- Eliminated “fs/direct-io.c” from target list of compiling
- blockdev_direct_IO(), blockdev_direct_IO_no_locking() and generic_file_direct_IO() in direct-io.c were replaced to in-line function or macro which only to return ″-EINVAL”.

5) FILE_LOCKING = n (fslock.patch) : 3.5%

- POSIX file lock function set (locks_mandatory_locked(), locks_verify_truncate() etc.) were replaced to empty functions.

- Evaluated the effect norming the size difference between vanilla-size and tiny as 100%.

5.3 Effective settings and patches to minimize Kernel size

7) BUG = n (nobug.patch) : 3.0%
   - BUG(), BUG_ON(), WARN_ON() and PAGE_BUG() were replaced to empty macro.

8) IGMP = n (igmp.patch) : 2.0%
   - net/ipv4/igmp.c was removed from the compiling target.
   - igmp related functions such as igmp_rcv(), ip_mc_join_group() etc. were replaced to empty macros.

9) POSIX_TIMERS = n (posix-timers.patch) : 2.0%
   - Most of kernel/posix-timers.c was removed. exit_itimers() and clock_was_set() were replaced to empty functions. do_posix_clock_monotonic_gettime() was modified to return “nano Second” based on jiffies.

- Evaluated the effect norming the size difference between vanilla-size and tiny as 100%.

5.3 Effective settings and patches to minimize Kernel size

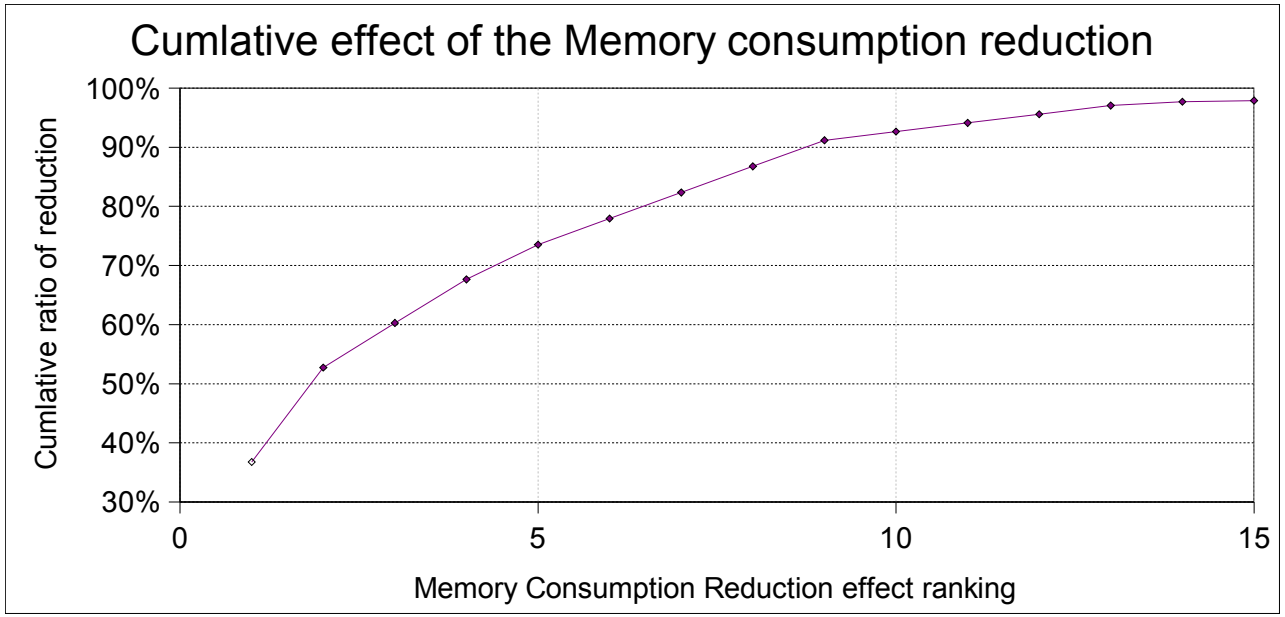
10)FULL_BUG = n (tiny-bug.patch) : 2.0%

- Removed _FILE_ and _LINE_ BUG() from macro definition in WARN_ON().

- Evaluated the effect norming the size difference between vanilla-size and tiny as 100%.

5.4 Effective settings and patches to minimize Memory Consumption

- 75% of the memory consumption reduction is performed by top 5 settings (patches). (evaluated the effect norming the size difference between vanilla-size and tiny as 100%).
  - Vmlinux : 74% of the effect brought by the top 5 settings
  - bzImage : 93% of the effect brought by the top 10 settings



5.4 Effective settings and patches to minimize Memory Consumption

1) MEMPOOL = n (mempool-shrink.patch) : 36.8%

- Reserve only one element in mempool_create(). Substitute pointer in mempool_t::cache.
  - ✔ In vanilla, it reserve the elements appointed by the parameter and substitute pointer into pointer array pointed by mempool::elements.
- Reserving the element trial failed in mempool_alloc(), it returns one element which was reserved and substitute Null into mempool_t::cache.
  It failed again in the next call of the function, it returns NULL.
  - ✔ In vanilla, when it fail to reserve element;
    - In case there are sufficient elements remaining in memory pool (more than 50% of the pool capacity), from the memory pool
    - In case not, it evoke bdflush Kernel thread and repeat until it can be reserved.

✗ Dead lock risk may be there in the highly loaded case.

- Evaluated the effect norming the size difference between vanilla-size and tiny as 100%.

5.4 Effective settings and patches to minimize Memory Consumption

2) SLOB = y, SLAB = n (slob.patch) : 16.0%

- Replace SLAB allocator to SLOB(Simple List Of Blocks) allocator.
- SLOB allocator does not wait SLAB object.
  For memory requests by kmalloc()/kmem_cache_alloc(), it returns the pointer after finding empty region in each request.
- The empty region is managed by single direction link of slob_t type structure. slob_t structure owns memory region size and pointer pointing the next empty region as the member.
- slob_t type structure is positioned at the top of the region.
  kmalloc()/kmem_cache_alloc() are to call slob_alloc(), obtain slob_t structure pointer, p, and return (void*)(p+1).

✗ Demerit: In case reserving multiple smaller memory regions than the page size, the efficiency of memory usage is high, however the concern of the fragmentation will be there when repeating the reserve and release.

✗ Demerit: Concern for the reduced system performance will be there because it retrieves the link in each reserve and release action.

✗ Demerit: Some settings SLOB allocator for Kernel 2.6.10 prevented Kernel booting up. SLOB for 2.6.11 works fine. (We could not yet resolved the reason.)

■ Evaluated the effect norming the size difference between vanilla-size and tiny as 100%.

5.4 Effective settings and patches to minimize Memory Consumption

3) AIO = n (no-aio.patch) : 7.6%

- Asynchronous I/O related function set (aio_compete(), exit_aio(), kick_iocb() etc.) in fs/aio.c are null defined.

4) DIRECTIO = n (direct-io-core.patch) : 7.4%

- See “Effective settings and patches to minimize Kernel size" item 5).

5) IGMP = n (igmp.patch) : 5.9%

- See “Effective settings and patches to minimize Kernel size" item 8).

- Evaluated the effect norming the size difference between vanilla-size and tiny as 100%.

5.5. Other (doubtful?) Settings and Patches

- Items which will not so much contribute the Kernel Size Reduction or Memory consumption reduction, while they may degrade the Kernel feature or the performance.
  - Change in-line functions to non-in-line functions
    - fs/inode.c ... ifind() (inode-inlines.patch) : 0.2%/-
    - fs/namei.c ... __vfs_follow_link() (namei-inlines.patch) : 0.0%/0.2%
    - include/arch/i386/kernel/semaphore.h ... up(), down() (semaphore-inline.patch) : 0.0%/-

    etc...

    ✗ If apply this modification on semaphore related functions, system doesn't boot up. (Both 2.6.10 and 2.6.11)
  - Delete API
    - ptrace (ptrace.patch) : 0.9%/
    - vm86 (remove-vm86.patch) : -/-
- Evaluated the effect norming the bzimage size difference between vanilla-size and tiny as 100%.

6. What's to do next?

- Evaluation on the embedded system platform
  - Porting to ARM platform
    - Number of patches under arch : 40 patches
  - Measurement of size and used memory
  - Benchmarking
    - With or without mempool
    - SLAB/SLOB allocator performance evaluation
    - Compare SLAB/SLOB allocator memory usage efficiency
- Development of more effective methods
  - Smaller in size
  - Less memory consumption
  - Higher performance
  - Maintain trace-ability
- Collaboration with CELF SystemSize W/G