Kernel Boot-Time Optimization

Nicholas Mc Guire
Distributed & Embedded Systems Lab
Lanzhou Universtiy, P.R.China
dslab.lzu.edu.cn

Funded by Siemens CTSE2 under contract FMU654213

Schedule

- Starting Point
- Problem statement
- Proposed Therapy
- Results and Conclusion

This study was funded by Siemens AG, CTSE2 Muenchen - we would like to thank them for there support of free-software !

Approach taken

- top down - source code analysis
- brute force - measure and hack
- strategic - analysis of the boot process concepts

This study was limited from `start_kernel` to `run_init_process`

top down approach

Tools for the top down approach

- KDB - Kernel Debugger (x86 only) - detailed code flow
- KFI - Kernel Function Instrumentation - function granular code patch
- Kernel GCOV - for code coverage in kernel space - global code flow overview
- UML - User Mode Linux testbeds - system scope

conducted on x86 (AMD 2GHz)

brute force attack

Tools used for the brute force attack

- instrument printk - timestamping on all printk
- GPIO signal and scope - hard coded "outb"
- hard-coded timestamping in code dumped via `/proc/timestamps`
- expect scripts (equivalent to instrument printk)

conducted on PPC405 (133MHz)

Strategic approach tools

- Kernel GCOV - for branch optimization
- LTT - Linux Trace Toolkit - post init
- Oprofile - Profiling with hardware support - detection of low level effects

Code size, code locality and related hardware artifacts need further investigation.

kernel boot-opt options

- resource stripping
- delayed allocation and initialization
- replace probing by config-options
- conceptual correction of the boot process

known kernel related hot-spots

- resource initialization
- console (printk)
- calibration
- root fs mount
- mount of /proc

One "new" hot-spot

Primarily we were interested in applying the available optimizations, this revealed a further essential boot time limit in the cache initialization.

- **memory initialization** - mem_init
- **cache setup** - fs/dcache.c: dcache_init,inode_init,mnt_init, etc.
- **therapy** -> initialize smaller caches - they will grow at runtime with hardly detectable overhead

Linux is very sensitive to low memory situations, deferring memory load where possible improves overall boot time behavior.

resource initialization

Most of the kernel boot time is obviously used for resource initialization - these need to be fit to the embedded environment

- reduce number of ramdisks/console/floppies/etc.
- reduce number of supported devices to a configurable minimum (do_mounts)
- shrink initial cache sizes to a minimum (fs/dcache.c)
- reduce number of inodes on filesystems (i.e. ext2)

console (printk)

- printk level initialized to "panic only"
- reduce the overall printk usage
- reduce the length of printk messages

calibration

Calibration does not really make much sense on every boot - but be ware that you need to sanity check the system in some way if using hard-coded values !

- lpj - preset Loops Per Jiffies
- use preset TSC quotient
- skip Update-In-Progress (UIP) wait (x86)

root-fs

Use a combination of filesystems to reduced the perception of the root-fs mount times - defers read-write to the fates possible point - after application launch.

- core in small cramfs - $<$ 20ms mount time
- use small ramdisk for early read-write
- mount mini_fo fore write access into initial ramdisk/pramfs
- mount jffs2 for general read-write access (as late as possible)

mounting /proc

Mounting proc is slow !

- reduce proc usage to a minimum
- put the interface into loadable modules (instead of `fs/proc/`)
- use sysctl (but not to extensively...)
- redesign init scripts so that `proc` is not needed at an early point

proc file initialization is very heavy weight - it would be nice if `/proc` could be loaded as a module.

Kernel boot time optimization - Results

- ppc45 133MHz 32MB RAM 4 Flash - start_kernel to prompt 188ms
- AMD K7 1GHz 256MB RAM 80GB HD - start_kernel to prompt 84ms
- cache initialization times reduced by 8ms (AMD K7)

It is surprising that a 133MHz system exhibits only roughly twice the boot time of a x86 PC system ! This indicates that there is still a not-identified bottle-neck involved at least on the AMD.

Cache init optimization - Results

- mem initialization times reduced from 57ms to 8ms on AMD by limiting memory availability
- mount times rise by 12 ms for ext2
- cache initialization times reduced by 8ms without memory constraints

Currently it is not posible to add physical ram into the memory pool of Linux after system initialization - modifications of Linux to allow this are under development though.

Conclusion

- Boot-time optimization related tools are available in a sufficient quality to allow customization of the boot-process
- With fairly moderate changes to the kernel very reasonable kernel boot times can be achieved.
- Be aware that most optimizations open the door to subtle boot-time failure modes that need to be assessed during development.

On our TODO List

Small kernel boot times are nice - but consumer electronics are interested in power-on-to-application-launch times, so the two main topics we hope to continue work on are:

- Boot loader optimization
- User-Space initialization optimization

Final Provocation

Linux is too fat

Is the unified "one-fits-all" approach of mainstream Linux really suitable for embedded systems and consumer electronics or is Linux not simply getting too fat ?

RTLWS8 Invitation

8th Real Time Linux Workshop - October 12-15 2006

Realtime Linux Foundation Inc.

held at Lanzhou Univiersity, Lanzhou, P.R.China

`http://www.realtimelinuxfoundation.org`