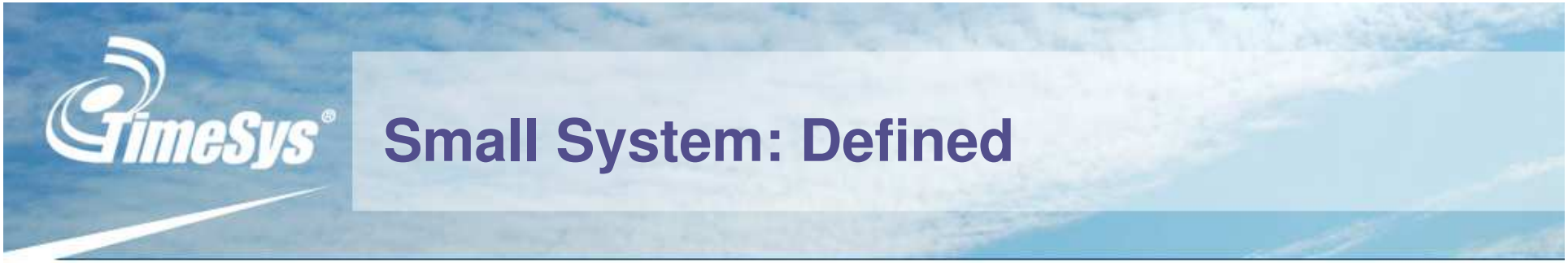
Making Linux Small

Presented by: Gene Sally

CELF Conference

Today's Topics

- Fitting Linux in a small system
  - Reducing the size of your RFS
    - Starting off right
    - Tools of the trade
  - Reducing size of kernel
    - Cull out what's not necessary
    - Small!= Low Functionality
- Approaches configuring your RFS
  - What libc to pick, if any
  - Removing deadweight from RFS
  - Space saving file systems



Small System: Defined

- **Memory**
  - Non-volatile: 4 MB
  - Volatile RAM: 4 MB
- **Processor**
  - ARM-based
  - Eden (x86 Licensee)
- **Devices Like This**
  - GumStix 64 MB RAM (I guess that's not so small)
  - PicoTux 2 - 4 MB RAM
  - Kontron E2Brain: 1 - 16 MB

A lot larger than a small system from a few years back ☺

Small RFS: Tools of the trade

- **Busybox busybox.net**

  A multi-call binary providing reduced-functionality implementation of most gnu command-line tools

- **uClibc uclibc.org**

  Alternate C standard library optimized for size, busybox symboit

- **Dietlibc www.fefe.de/dietlibc**

  Another alternate library optimized for reduced size, supports only static linking

- **strip**

  Just running strip to ren symbols can greatly re

There's no magic tool! Each system requires thought and experimentation, but these tools can quickly get you started down the right path.

Small RFS Rules of Thumb

Minimal

Glibc and Busybox

6 - 10 MB

- Strip glibc libraries and remove locale data
- Use busybox to supply user-land RFS programs

Small

uClibc and Busybox

2 - 2.5 MB

- Use the uClibc as your C library, recompile applications as well
- Busybox for RFS
- Manually create device nodes

Tiny

Paired-down uclibc and BusyBox

1.5 - 2 MB

- Put effort into reducing what's included in uclibc and busybox.
- Write your own system initialization scripts

Minuscule

No BusyBox (or very minimal) no shared libraries

<1 MB

- If you run one application, this is a great solution
- Minimal functionality!

Minuscule File System: How is that possible?

- Static linking removes the need for a C library
  - Even a small shared library has a lot of code that's never called
  - Duplicate code < unused code, so you save space
- Change the default init
  - Create your own file or binary named init (initrc)
  - Init=<your program here>
- Create all device nodes in advance
  - For almost all embedded systems, the devices remain fixed
  - Reduces memory requirements and boot time
- Drawbacks
  - If device has several programs, the libraries + programs will probably be smaller
  - Not using init means a doing re-spawn for your application and custom start-up code, the kernel does not take kindly to init terminating
- Counter-point
  - System considerably simpler
  - Much greater control over start-up process means better performance
  - Fewer resources necessary for start-up

Beyond BusyBox

- **BusyBox**

  provides a great starting point for an embedded system but there's still a wealth of packages for embedded systems.

- **Dropbear**

  ssh/scp implementation

- **Boa/thhpd**

  Small & fast http servers

- **Stupid-ftp**

  A small & simple ftp server

- **Zebra**

  IP protocols: RFC 1771 RIP

- **ACE/TAO**

  Corba on your device

Space Efficient File Systems

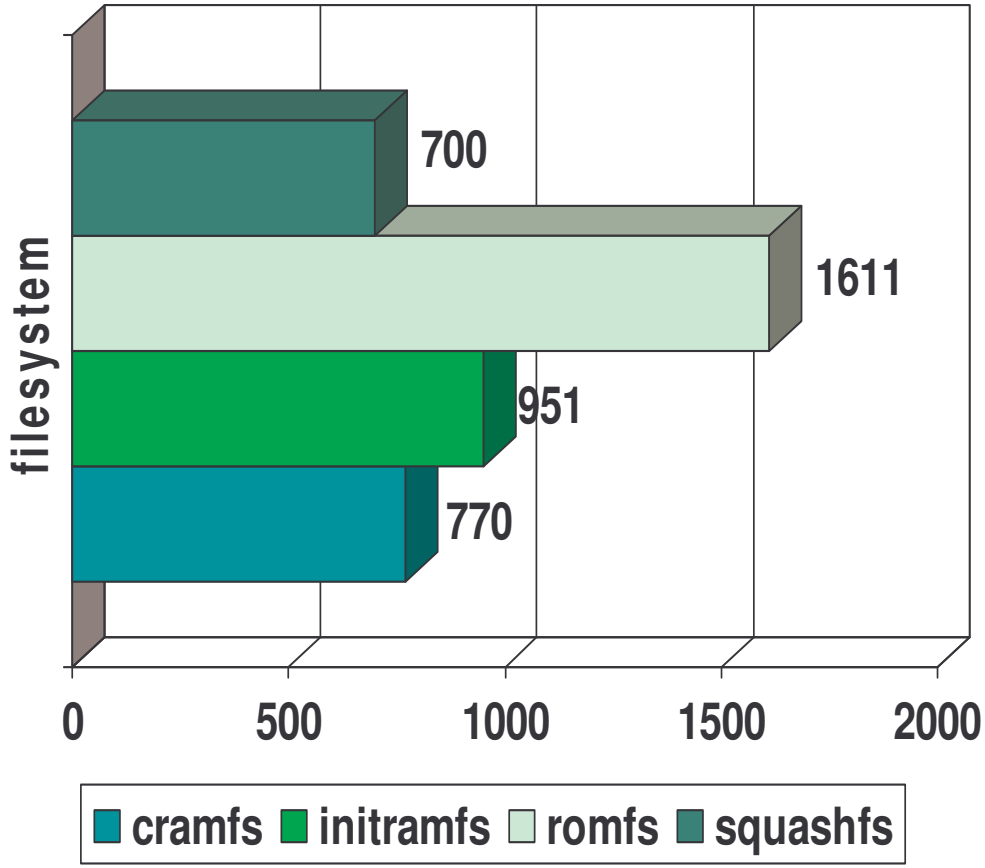
- Most use compression schemes that result in an even smaller file system image
- Most are read-only
  - Create RAM drives for temporary storage needs
  - Use another partition for read\write file system
- Alternatives to ext3 or jffs2
- Bake-off, which produces the smallest RFS?
  - Used host tools to create file system image for each
  - Compare final size of each
  - Smallest != Best. Fast, small, low-memory: pick two.

Drive By: Space Efficient File Systems

cramfs ▪Compresses files to save space ▪Meta-data not compressed, but stored efficiently ▪Performance hit when reading data, as it must be decompressed	▪Files < 16MB ▪Maximum filesystem size 256 MB ▪Does not store timestamps ▪Write-only
romfs ▪Simple and fast ▪Driver does not need overhead of compression library ▪Fast access times	▪No last access, uid/gid information stored ▪Read-only ▪No compression
initramfs ▪Well supported for 2.6, low kernel overhead ▪Records all meta-data, space efficient ▪Easy to create file system	▪Uid/gid stored only as numbers ▪Support outside of 2.6 spotty ▪Will grow unbounded, until exhausting memory
squashfs ▪Uid/gid stored only as numbers ▪Support outside of 2.6 spotty ▪Will grow unbounded, until exhausting memory	▪Read-only ▪Slower access time due to decompression overhead

Tiny RFS: Bake-off results

- Starting size: 2.1 MB
- All solutions produce "small" root file systems
- squashfs produced the smallest RFS size, but has a little more overhead in the kernel
- Depending on the nature of your file system, your results may vary



Other Suggestions

- Use RAM based file systems for temporary data
  - No need to reserve parts of flash for data you don't care about
  - You can use RAM disks or tmpfs file systems
- Pre-create as much as possible
  - Create directories and files in advance
  - Create device nodes in advance
  - Remember, shell scripts require a shell, and that takes space.
- Test your file s
  - While not per

One more thing: it's easier/faster if you work up from 0 rather than down from the default RFS size. This tactic forces you to think about every byte.

Making the Kernel Smaller

- Kernel, in default state, targeted for desktop systems
  - Device flexibility
  - Start-up time not that important (can afford the time delays related to device discovery)
  - Robust networking support
  - Support for a wide variety of input devices
  - System doesn't know what it could be running
- Embedded environment
  - Fixed set of peripherals (most of the time)
  - Fixed purpose
  - Start-up time very important

Minimizing the kernel

- Remove debugging information & kernel hacking
- Don't load modules
  - Compile them directly into the kernel
  - Can remove support for modules (bonus: the RFS doesn't need insmod either)
- Other low-hanging fruit
  - Drop video support
  - Chances are you won't need support for IDE
  - Compile with -Os (makes debugging harder)
  - Remove suppo

For those interested in Matt Mackall's "Linux-Tiny" patches, they've been (mostly) rolled into the 2.6 line.

Breaking News: Linux-Tiny Patches

- "Seems not to have been maintained after kernel version 2.6.14" http://tree.celinuxforum.org/CelfPubWiki/LinuxTiny
- Still available at CELF for the 2.6.16.19 kernel
  - Or at http://www.selenic.com/tiny/
  - The patches are (thoughtfully) separated by specific functionality, or in one large patch
  - Interested users can get these patches to apply on their kernel. Considering the size and content, this shouldn't be too hard…

Kernel Minimization (continued)

- Don’t support dhcp\bootp if you’re not using your network device frequently (or at all in production)
- Remove sysctl if you don’t need to dynamically change your kernel configuration
- Remove ext2/3 support of you’re not using this file system, most configurations include this by default
- Virtual memory, if the idea of a swap drive in RAM isn’t funny enough, consider it on a flash device.
- Reduce the kernel log ring buffer (default is 16K)
- Drop hot-plug support and Kernel User Space Events

What was removed

- **Drop**

  all file systems except the one's you're using! Ext3 support is 133K alone! If you have a ram disk, using ext2 is fine.

- **CONFIG_SWAP (30K)**

  Swap devices

- **CONFIG_SYSCTL (23k)**

  Change kernel parameters on the fly

- **CONFIG_LOG_BUF_SHIFT**

  Reduce to 4K

- **CONFIG_HOTPLUG (10K)**

  Support for PCMCIA devices

- **CONFIG_KOBJECT_UEVENT (3.5k)**

  Kernel object notification to user space programs, related to CONFIG_HOTPLUG.

- **CONFIG_EMBEDDED**

  Enable this, but under the options, disable CONFIG_KALLSYMS

- **CONFIG_MODULES**

  Support for modules

- **CONFIG_BINFMT_AOUT (7K)**

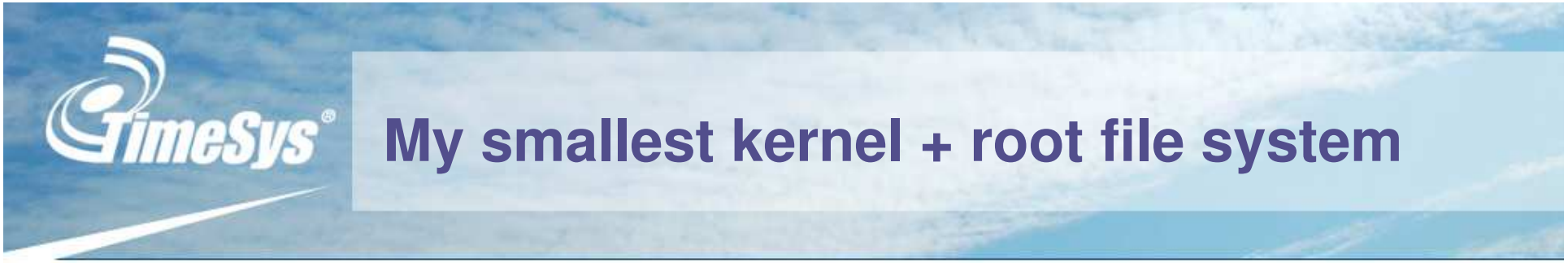
  Support only ELF executables (the standard since 1995)

- **CONFIG_MD (45K)**

  Raid devices? Probably not for you. Raid + jffs2 == funny!

- **CONFIG_DEBUG_KERNEL (not sure)**

  When deployed, this information can be used.



My smallest kernel + root file system

Kernel (bugboot image)	2,100 KB
Root File System	700 KB
Total	2,800 KB

The RFS could be smaller yet by reducing the features in busybox and uClibc. Bugboot seems to be a bit larger than other formats.

Lots of URLs in today's presentation

- **Busybox**
  http://busybox.net
- **uClibc**
  http://uclibc.org
- **Dietlibc**
  http://www.fefe.de/dietlibc
- **Cramfs**
  http://sourceforge.net/projects/cramfs/
- **romfs**
  http://sourceforge.net/projects/romfs/
- **initramfs**
  http://packages.debian.org/unstable/utils/initramfs-tools
- **squashfs**
  http://sourceforge.net/projects/squashfs

Q & A

Thank you for attending!

Drop by www.timesys.com.