

Real Time Linux in Embedded Systems

Nicholas Mc Guire
der.herr@hofr.at
FSMLabs Data GmbH - AUSTRIA

Schedule

- Starting point
- Features of MiniRTL
- Strategies for embedding RTLinux
- RTLinux example system and app
- Security considerations
- Availability
- Acknowledgement

Slide 1

Slide 2

Starting point

Embedded system development is moving away from single-task/user applications running on dedicated target platforms towards a reduced general purpose OS running on PC-like hardware. MiniRTL offers:

- bootability off a 1.44MB media (currently simply a floppy)
- hard-real time capabilities, based on FSMLabs' current RTLinux development
- maximum compatibility to a "standard" desktop development system running an up-to-date kernel

In this paper strategies to reach these goals will be described and the successful implementation in a running system, MiniRTL, shown.

Slide 3

Why Linux ?

Traditionally, system software for embedded systems has been developed independently of general purpose operating systems. So why take mainstream Linux as a basis and drop it to an embedded system?

- GNU/Linux is a stable desktop development system.
- Embedded Linux is compatible to a desk-top system
- No special "tools" are required for development

Slide 4

System Overview

MiniRTL offers an up to date Linux system.

- Up to date Linux Kernels 2.2.14 (2.2.17 is at work)
- Current RTLinux versions rtlinux-V2.3
- Glibc 2.0.7
- Support for most current hardware.

Moving up to Glibc 2.1.9 Kernel 2.2.17 is in progress at time of writing.

Slide 5

Network Features

The network capabilities of MiniRTL are not specific to MiniRTL and are listed here only because they might not have been taken into account when considering embedded systems.

- All basic network protocols are supported (TCP/IP, UDP, ICMP,...)
- inetd offering telnet tftp and SMTP
- sshd giving secure access via ssh and scp
- mini_httpd, web-server with cgi-bin support
- central network services (DNS,DHCP,NTP,etc.)

MiniRTL can be treated as a full-featured Linux host in a networked environment.

Slide 6

System Services

- Cron for periodic administrative tasks
- Syslogd/Klogd for system monitoring
- Shell access at the console (ash)
- Standard system commands for administration and configuration

Slide 7

Add on Packages

MiniRTL is only a "core" system, providing basic system capabilities. Adding capabilities to the kernel and dropping in user-progs is simple.

- Network capabilities like NFS, Routed, Named are easily added as modules
- User programs including additional libs are simply a tar.gz archive
- The core system is provided in distinct packages, which can be modified or removed if not needed.

Slide 8

Design Guidelines

Embedded systems are low-memory systems. While designing an embedded Linux system, a few points need to be kept in mind.

- Linux is sensitive to low memory situations
- Ramdisks reside in buffer cache
- Dynamic disk requirements must be taken into account.

Doubling available memory on low-memory PC's can increase overall performance as much as doubling CPU speed !

Slide 9

Reducing System Size

Since memory footprint is so critical, strategies to minimize memory demand are the key to building embedded systems.

- Exploit redundancies
- Delete what is no longer required during runtime
- Compress files where long load times are acceptable
- Reduce runtime size of kernel/modules/executables
- What is only required for administration / configuration can be loaded on demand

Combining these strategies gives a stable easily administratable system running with a filesystem footprint of less than 4MB.

Slide 10

Exploiting Redundancy

Unix systems traditionally have many different ways to do one and the same job, think of cat, less, more, tail and even dd, all can be used for the same purpose. Decisions to make:

- What do we really need ?
- Are all options of the program required for our setup ?
- Can scripts replace some binaries ?
- Will a simpler service do - like replacing ftp by tftp ?
- What could be loaded at runtime ?

Slide 11

Deleting and Compressing

In desk-top systems, executables will be permanent. On ramdisk based systems, the lifetime of an executable is only until the next reboot. This opens additional possibilities:

- Kernel modules can be deleted after boot up
- Initialization scripts/programs may be deleted
- Applications and data can be compressed manually or using cron-jobs
- Applications should store data in compressed format
- Move data/logfiles off-site

Minimizing Runtime Size

Reducing runtime size of executables is performance critical. Many executables are designed for large systems, allocating memory adequate for large systems, but wasteful for an embedded system.

Slide 12

- Build kernel resources as modules
- Stripping out resources from the kernel (number of tty's, hd's, floppy formats etc.)
- Strip out options that are not required for your system from executables (who ever used all options of find or ls ?)
- Use multicall binaries where possible

Multicall binaries

Many standard desk-top binaries are multicall binaries like gzip/gunzip in smod/rmmod. This techniq can help reduce executable size for embedded systems.

Slide 13

- Only one ELF-header for many executables
- One common argument parser
- "Build-in libs" for a group of executables
- Reduce filesystem trace.

A prominent and impressive example of such a multicall binary is busybox .

Slide 14

The busybox concept

Busybox is a multicall binary that includes many standard system tools in a reduced form, resulting in a dramatic reduction of size.

- the programs in bb (68K) would ocupie around 600K with standard tools
- a "hello world" function increases busybox only by 161 bytes !
- busybox only requires a smal set of libs for operation
- simple configurability at compiletime allows selection of exactly your needs

Busybox was designed and written by Eric B. Anderson and Dave Cinege .

Slide 15

The minimum set up libs

Deciding on a set of libs to use is essential for such a system, libs are large and applications should be designed not to require any "exotic" libs. Ideally all apps should share a minimal common set. For MiniRTL this set is

ld-2.0.7.so	libc-2.0.7.so	libcrypt-2.0.7.so
libdl-2.0.7.so	libncurses.so.4	libnsl-2.0.7.so
libnss_db-2.0.7.so	libnss_dns-2.0.7.so	libnss_files-2.0.7.so
libresolv-2.0.7.so	libss.so.2.0	libutil-2.0.7.so
libuuid.so.1.1		

Slide 16

Reducing kernel resources

Memory footprint being so critical, much consideration should be given to the kernel's memory footprint. Starting points for resource stripping might be:

- Only enable support for the specific setup, e.g. reduce supported floppy formats to exactly the one you need.
- Strip out autoprobng where you can pass values on the commandline
- Reduce allocated resources, an embedded system does not need 64 consoles or eight IDE devices.
- Reduce options for filesystems (maximum filesize of 2GB namelength of 1024chars, directory debth etc.)

Slide 17

Reducing kernel resources: Warning

Be warned, cerless stripping of resources can result in loss of compatibility to the desk-top development system. Finding errors due to such modifications is a very painful and irritating sturgle.

Slide 18

Kernel Modifications in MiniRTL

The modifications listed here were not done as part of the MiniRTL project, but rather were the basis for this project.

- initrd-always kernel patch, ensuring the call of linuxrc at bootup
- mkminixfs kernel patch, to set up the ramdisk
- RTLinux from FSMLabs, which permits hard realtime processes on embedded systems

The initrd and mkminixfs kernel patches were taken from <http://www.psychosis.com>, the rtlinux kernel is maintained and developed by FSMLabs at <http://www.fsmlabs.com>.

Slide 19

initrd-always

The initrd-always kernel patch takes care of two problems for ramdisk based systems :

- It launches the configuration script linuxrc, which is a regular ash shell script, making maintenance and adapting easy.
- It sets up a pseudo console device (linuxrc.tty) since at that time linux doesn't yet have a console available.

Slide 20

mkminixfs untar

The mkminixfs kernel patch is responsible for:

- Setting up a minixfs on /dev/ram0.
- Adding the "untar" function to the kernel, allowing use of standard tar file archives as packages.
- Adding a "unzip" function allowing for decompression of standard GNU-gzip files.
- Setting the real root device correctly after initializing the newly created filesystem.

Slide 21

FSMLabs rtlinux V2.3

rtlinux is a kernel patch to allow for hard realtime operations on a Linux system. The basic concepts include:

- Unconditional interrupt interception, no interrupt directly reaches the Linux OS
- High resolution timing functions, giving nano-second resolution (limited by the hardware only)
- Independant scheduling instance, controlling the realtime tasks
- Running Linux as the lowest priority idle task, passing all interrupts destined for non-realtime tasks as "soft-interrupts"

These modifications permit guarantied worst case jitter as low as 15 microseconds on x86 platforms.

Slide 22

MiniRTL on DSP Design TB486

As a sample embedded system MiniRTL was deployed on the TB486 from DSP Design. This typical embedded system features:

- AMD Elan SC400 at 33MHz
- 8MB Flash-card, 2MB used for the system
- 16MB DRAM, 4MB used as RAMDISK at runtime
- All standard PC peripherals available (VGA, Ser(x3), Par, Ethernet, etc.)
- 16 channel 12bit analog in, 2 channel 12bit analog out, 24 channels digital I/O

More info on the TB486 is available at <http://www.dspdesign.com>

Slide 23

rtlinux-V3.0 on TB486

Running rtlinux-V3.0 Kernel 2.2.17 on the TB486 allows stable hard realtime operation. Giving

- 15 microseconds worst case jitter
- 10 microseconds event resolution
- 17 nano-seconds timer resolution
- 6 microseconds interrupt response time (hardware dependant, this value was measured on parallel port interrupts)

These results are from a system running at a load average between 25 and 30, with multiple interrupt sources active (Ethernet, PS/2 mouse, Keyboard, Serial port).

Slide 24

hello.o a simple realtime app

The first C-application is traditionally "hello.c" so the first realtime kernel module is most naturally "hello.o". This sample realtime application shows how simple a rt-thread is set up. The three basic functions of every kernel module:

- `init_module`, register the module & setup resources
- The actual task "say_hello"
- `cleanup_module`, release resources and unregister the module

Slide 25

hello.o : init_module

Init module is responsible for checking that the required resources are available and registering the module and its functions with the kernel.

```
int init_module(void) {
    return pthread_create (&thread, NULL,
        say_hello, 0);
}
```

Slide 26

hello.o : the "main" routine

```
void * say_hello(void *arg)
{
    struct sched_param p;
    p . sched_priority = 1;
    pthread_setschedparam (pthread_self(), SCHED_FIFO, &p);

    pthread_make_periodic_np (pthread_self(), gethrtime(), 500000000);

    while (1) {
        pthread_wait_np ();
        rtl_printf("I'm here; my arg is %x\n", (unsigned) arg);
    }
    return 0;
}
```

Slide 27

hello.o : cleanup_module

Cleanup module is responsible for unregistering all functions related to this module, freeing any allocated resources and reducing the module count for this module.

```
void cleanup_module(void) {
    pthread_delete_np (thread);
}
```

Slide 28

Final conclusion

This project successfully demonstrates a stable embedded real time Linux System. It shows that:

- Embedded RTLinux is an alternative to proprietary embedded RTOS.
- RTLinux is a stable, easy to use Linux extension adding reliable hard real time capabilities to embedded systems.
- An open source system can greatly simplify development and help reduce cost of embedded systems.
- Embedded Linux gives access to low cost hardware for embedded platforms.

Slide 29

Security in Embedded Systems

The upcoming discussion in embedded systems is not performance, it's not memory footprint, it's security !

- Network access for ease of diagnostic and administration
- Standardized protocols and software
- Simple to use interfaces, no cryptic config files.

Linux is a system that offers high security standards, but these must be taken into account from the first minute of design on.

Slide 30

Security Capabilities of Linux

Since Linux is deployed on millions of desk-top systems, the Linux security features have been well tested. These features include:

- Encrypted connections (SSL, VPN, etc)
- Reliable Password access (shadowing)
- Reliable File encryption

These mechanisms are not black-boxes: they are open standards and open-source software, you know what you are using.

Slide 31

Security Vulnerabilities of RTLinux

RTLinux has some security related problems, notably the requirement that users operating these systems need root-privileges to do so. This requires some consideration for design of embedded systems.

- Split active and passive operations cleanly
- Go for the lowest privileges for passive operations possible
- Move to off-site reporting where possible
- Keep privileged actions well documented
- Design logging around the security related items

Availability

This open source project is available on the internet for download at:

- FSMLabs.com,
<http://www.fsmlabs.com/> <ftp://ftp.fsmlabs.com>
- RTLinux.org,
<http://www.rtlinux.org/minirtl.html>
<ftp://ftp.rtlinux.org/pub/rtlinux/minirtl>
- Thinkingnerds.com,
<http://www.thinkingnerds.com/projects/minirtl/minirtl.html>
<ftp://ftp.thinkingnerds.com/pub/projects/minirtl/>

Slide 32

Slide 33

Comments and questions to der.herr@hofr.at

Slide 34

Project Supporters

- University of Vienna, Inst. for Material Science, Computational Material Science Group, Prof. Dr. J. Hafner.
<http://mpi.univie.ac.at>
- FSMLabs Inc., the RTLinux Company, Socorro New Mexico, USA
<http://www.fsmlabs.com>