Introduction to Embedded Linux

EUGLUG Presentation Series

Saturday September 24th 2005

Introduction to Embedded Linux

- What is an embedded computer?
- Why use Linux in an embedded System?
- Running Linux without an MMU.
- Running Linux on Multicore systems.
- Debugging embedded Linux.
- Toolchains and evaluation boards.

What is an embedded Computer?

1.) A device which has its own computing power dedicated to specific functions, usually consisting of a microprocessor and firmware. The computer becomes an integral part of the device as opposed to devices which are controlled by an independent, stand-alone computer. It implies software that integrates operating system and application functions.

What is an embedded Computer?

2.) A computer which is part of some product such as a car or plane, and is connected to sensors and actuators rather than to the usual keyboard, screen etc. Although such a computer runs only a single program, it is often better to use a cheap mass-produced general purpose processor than to develop a special-purpose chip.

What is an embedded Computer?

3.) An embedded system is a special-purpose computer system, which is completely encapsulated by the device it controls. An embedded system has specific requirements and performs pre-defined tasks, unlike a general-purpose personal computer.

Examples

Networking Equipment

PDAs

Cellular Phones

Test Equipment

Home Appliances

Automotive Engine control & others

Medical Equipment

Numbers

- Some modern Vehicles can have ~70 Computers embedded in them.
- Typical American home has >40 Embedded processors
- Embedded Computers account for >99% of all CPUs used.

What Makes Embedded computing Different?

- Designed for Single (or few) function(s) at lowest cost
- Often Require very long (years) continuous uptime
- Often Have Real Time constraints
- Usually require less power use
- Often Lack typical computer peripherals
- More often than not Embedded computers use non x86 architectures.

Why use Linux?

- Embedded Systems often require extensive customization. This is Easy with Linux because it is open source.
- Linux is Free. No Licensing Costs, no per unit shipped royalties.
- Linux can be stripped down to be very lightweight. (complete router in less than 2MB flash 8MB RAM)
- Linux is already ported to a wide variety of Architectures. (x86, ARM, MIPS, XScale, SuperH, PowerPC, 68k 29k)

Why use Linux?

- There already exists a large body of engineers and programmers familiar with the Linux environment
- Linux is stable. Server uptimes routinely hit months if not years.
- There exists a large body of functional applications for linux which can be quickly adapted for your project rather than rewriting fhem from scratch.

Linux without an MMU

What does Linux use an MMU for anyway?

Linux uses an MMU to:

- Switch Processes Quickly
- Implement Copy on write Paging. (speeds up forking)
- Share code segments between processes while allowing different data segments (IE Shared libraries)
- Allocate new stack space to a process

Linux without an MMU

Why does this Matter?

Without an MMU it is very difficult to:

- Switch Processes Quickly
- Fork new processes.
- Share code segments between processes
- allocate new stack space to a process

Linux without an MMU

What the heck is an MMU?

MMU == MEMORY MANAGEMENT UNIT

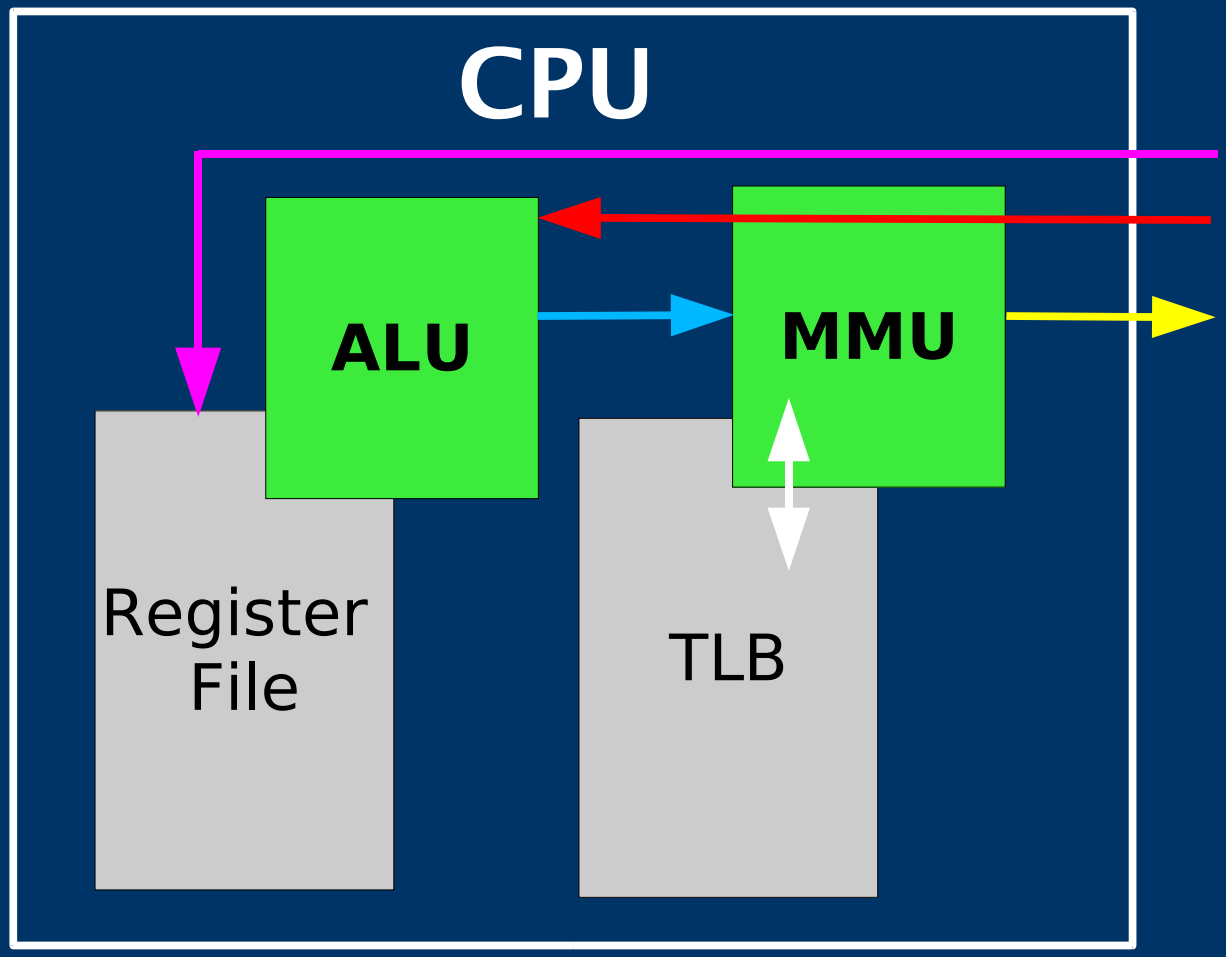
Linux without an MMU

How does it work?

Typically part of the CPU. It has a small amount of memory which it uses to hold a table which matches “virtual addresses” to “physical addresses” in RAM. This is known as the “Translation Look-aside Buffer” or TLB. Anytime the CPU needs to read RAM the request is sent to the MMU. The MMU matches the requested virtual address with the actual physical location of the requested data in external memory.

Linux without an MMU

Hunh?



CPU receives instruction to fetch data at address 0x00100000 into register r3.

ALU issues Read request

MMU looks in TLB and determines that this is really 0x8f100000 in RAM

MMU issues the read to the external memory system.

CPU receives data from 0x8f100000 and proceeds with the next instruction.

Linux without an MMU

How do you work around these limitations?

use uClinux (you-see-linux)

www.uclinux.org

www.ucdot.org

Linux without an MMU

What are the differences between uClinux and normal Desktop linux

- uClinux doesn't impliment fork
- uClinux can't autogrow stack (application stacksize must be set at compile time)
- uClinux doesn't have memory protection
- uClinux uses uClibc not glibc
- uClinux doesn't support shared libraries

Multicore embedded Linux

Why might you want to run Linux on multiple cores in a single product?

To allow some tasks hard real time priority

To parallelize certain tasks

Division of Labor. 2 smaller less expensive specialized cores often can do the job better than one faster multipurpose one.

Multicore embedded Linux

SMP or seperate OS instances?

Seperate OS instances

Cell phones

VoIP

Video Conferencing Equipment

SMP

Routers

VPN boxes

Multicore embedded Linux

Challenges of Multicore embedded linux

How do you decide how to split up the work?

Building effective multithreaded applications with limited tools/libraries.

As the number of cores in the design goes up the performance increase per core goes down.

Debugging on multiple cores is difficult.

Debugging Embedded Linux

Methods of debugging

Printf ( ) [and its cousin printk( )]

core dumps

gdb stubs

kgdb

ICE (In Circuit Emulation)

Debugging Embedded Linux

printf( ) and printk( )

Everyone has used these.

Easy to instrument code and turn on and off debug output.

Not hard real time information (usually console output is done by a seperate task)

Doesn't help if your kernel crashes before it can get to the point of outputting any text.

Debugging Embedded Linux

core dumps

Standardized

Hard to use

Often can be misleading (esp in the case of one task corrupting the memory space of another)

Debugging Embedded Linux

gdb stubs

Allows use of GDB to debug embedded linux applications.

Provides best application level debug short of porting GDB.

Must be ported to target

Can affect timing and disturb the outcome of race condition bugs.

Requires an interface to GDB (serial, Ethernet, USB)

Debugging Embedded Linux

kgdb

Allows use of GDB to debug embedded linux kernel.

Must be ported to target.

Can affect timing and disturb the outcome of race condition bugs.

Requires an interface to GDB (serial, Ethernet, USB)

Debugging Embedded Linux

In Circuit Emulation -- JTAG

Allows use of GDB or third party debugger application to debug embedded linux kernel.

No porting required.

Effective even very early in the boot process.

Does not affect timing.

Costs money. Up to $5000 in some cases for the tools.

Your processor may or may not support the hardware needed.

Getting started with Embedded Linux

Obtaining an evaluation board

VersaLogic -- x86	www.versalogic.com/Products
Gumstix -- XScale	www.gumstix.com
KwikByte -- ARM9	www.kwikbyte.com
MicroTik -- MIPS 4kc	www.routerboard.com
Linksys wrt54g - MIPS	www.linksys.com

Getting started with Embedded Linux

Obtaining a toolchain

I recommend finding an eval board with the toolchain included and the kernel preported. this will save a LOT of time and frustration. Especially if you are very familiar with Linux Kernel Internals.

Otherwise:

ARM: www.gnuarm.com precompiled cross toolchain
http://frank.harvard.edu/~coldwell/toolchain/
cross compiler build instructions

x86: Normal gnu tools work fine. too many options to list

MIPS: http://www.linux-mips.org/wiki/MIPS_SDE_Installation

cross tools & native tools instructions & downloads

Getting started with Embedded Linux

Websites with useful info for Embedded Linux

www.linuxdevices.com – general embedded Linux news eval board listings

www.linux-mips.org – Linux on MIPS processors

www.uclibc.org – smaller sized replacement for glibc

www.ucdot.org – Embedded Linux Developer Forum

Getting started with Embedded Linux

Commercial Embedded Linux providers

MontaVista Linux – www.mvista.com

Jungo Software – www.jungo.com

Timesys Linux – www.timesys.com

WindRiver – www.windriver.com – Also VxWorks

Introduction to Embedded Linux

END

Mike Cherba
mcherba@gmail.com
www.euglug.org