

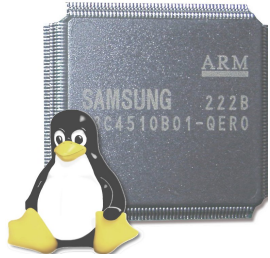
Do relógio de pulso a clusters NUMA: Portando o Linux para novas arquiteturas

Felipe Wilhelms Damasio

Lucas Correia Villa Real

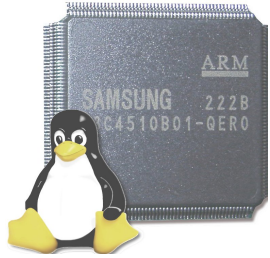
Ricardo Nabinger Sanchez

7º Fórum Internacional de Software Livre



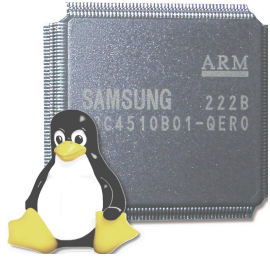
Conteúdo

- Apresentação
- Arquiteturas Desktop vs. Embarcadas
- ARM Linux
- Portando o Linux para uma máquina ARM
- Timers e seus efeitos colaterais
- Depuração do kernel
- Considerações finais



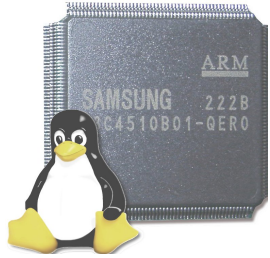
Apresentação

- Sistemas embarcados
 - Arquiteturas dedicadas / portáteis
- Exemplos:
 - Televisores, palmtops, vídeo-games, máquinas de cartão de crédito, ...
- Arquiteturas embarcadas
 - PowerPC, ARM, MIPS, SH, ...



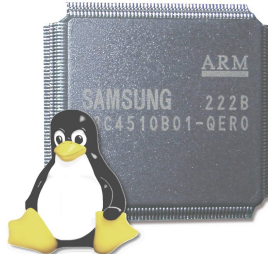
Arquitetura: Diferenças

- Layout de hardware “fixo”
- System-on-a-chip (SoC)
 - Controladores USB, DMA, UART, IRQ, LCD, A/D, RTC, Power Management, ...
- Componentes não expansíveis
 - Memória
 - Processador



Arquitetura: Diferenças

- Armazenamento de dados
 - RAM/ROM, Flash chips, memory cards
- Processadores
 - MMU / MMU-less, FPU / FPU-less
 - RISC, clocks modestos
 - Big/Little endian
- Componentes
 - RTC: minuto/hora/dia/mês/ano
 - LCD: pode ter orientação diferente da esperada

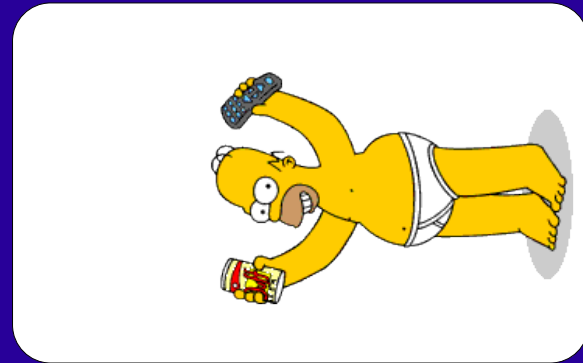


Arquitetura: Diferenças



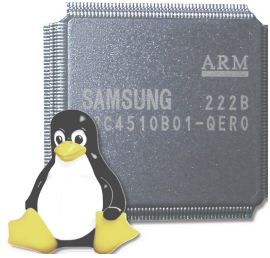
Videogame Sem Nome

Orientação Esperada



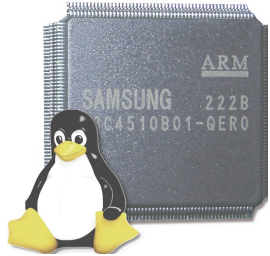
Videogame Sem Nome

Orientação Obtida



ARM Linux

- Port do Linux para máquinas baseadas em ARM
- Suporte para mais de **700** máquinas
- Integradas no kernel ou em projetos paralelos
 - `linux/arch/arm`
 - <http://www.handhelds.org/>

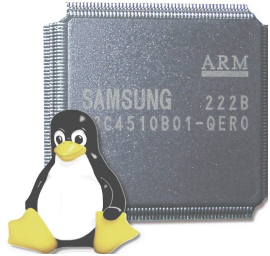


ARM Linux

- Diferentes plataformas são suportadas no kernel

```
linux-2.6.16-git18/arch/arm$ ls -d mach*
```

```
mach-aaec2000      mach-footbridge  mach-ixp2000     mach-omap2
mach-sa1100        mach-at91rm9200  mach-h720x       mach-ixp4xx
mach-pxa           mach-shark        mach-clps711x    mach-imx
mach-l7200         mach-realview    mach-versatile   mach-clps7500
mach-integrator    mach-lh7a40x     mach-rpc          mach-ebsa110
mach-iop3xx        mach-omap1        mach-s3c2410
```



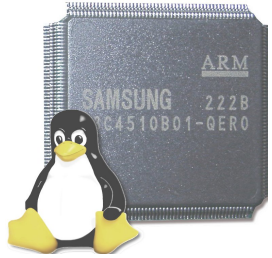
ARM Linux

- Diversas máquinas são implementadas com base nessas plataformas

```
linux-2.6.16-git18/arch/arm$ cd mach-s3c2410
```

```
linux-2.6.16-git18/arch/arm/mach-s3c2410$ ls mach*
```

```
mach-anubis.c      mach-h1940.c      mach-nexcoder.c  mach-rx3715.c
mach-smdk2440.c   mach-bast.c       mach-n30.c       mach-otom.c
mach-smdk2410.c   mach-vr1000.c
```

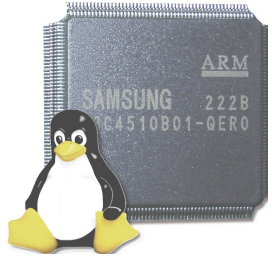


ARM Linux: Estrutura de Diretórios

linux-2.6.16-git18/

arch/arm/

boot/	inicialização (mmu, cache, page tables)
configs/	.config default para diferentes plataformas
kernel/	API usada na árvore do ARM
lib/	funções (putuser, setbit, memset, strchr)
mach-*/	implementações de máquinas
mm/	manipulação de cache, TLB, page fault, DMA
nwfpe/	Netwinder floating point emulator
tools/	database de máquinas ARM registradas



ARM Linux: Estrutura de Diretórios

linux-2.6.16-git18/

include/

asm-arm/

arch-*/

hardware/

mach/

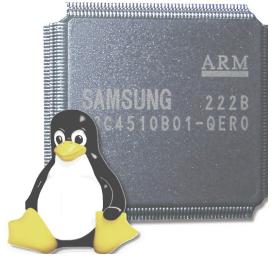
linux/

registradores, memory map, canais dma

API de clock, macros para I/O

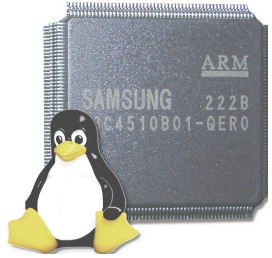
API de DMA, flash, IrDA, IRQ, PCI, timers

protótipos para drivers e sub-sistemas



Portando o Linux para uma máquina ARM: requisitos

- Especificações do hardware
- Estudar o código e a especificação de alguma arquitetura similar já portada
- Ter um cross-compiler preparado
 - <http://www.codesourcery.com/>
 - <http://www.kegel.com/crosstool>



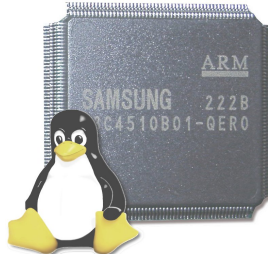
Portando o Linux para uma máquina ARM: requisitos

- Registro da máquina: permite tratamento diferenciado no kernel

<http://www.arm.linux.org.uk/machines>

- Máquina ganha entrada em [arch/arm/tools/mach-types](#):

# machine_is_xxx	CONFIG_XXX	MACH_TYPE_xx	núm.
Roadrunner	MACH_ROADRUNNER	ROADRUNNER	704
at91rm9200ek	MACH_AT91RM9200EK	AT91RM9200EK	705
gp32	MACH_GP32	GP32	706



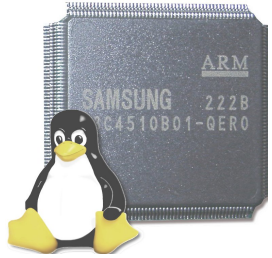
Portando o Linux para uma máquina ARM: requisitos

- `mach_types` provê variável de configuração para a máquina:

```
#ifdef CONFIG_MACH_GP32
...
#endif
```

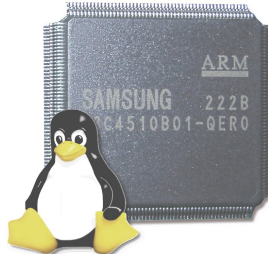
- E verificação em tempo de execução:

```
if (machine_is_gp32()) {
    ...
}
```



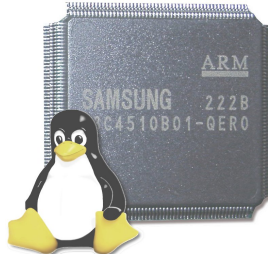
Portando o Linux para uma máquina ARM: bootloader

- Configura e inicializa a RAM
- Configura uma porta serial
- Desabilita DMA, MMU, Caches (dados e instruções)
- Detecta o tipo da máquina
- Configura uma lista de informações para o kernel
 - memória/initrd (início + tamanho)
 - cmdline
- Chama a imagem do kernel, passando ID da máquina



Portando o Linux para uma máquina ARM: kernel

- Primeiro passo: **detecção da CPU**
 - O kernel mantém uma tabela das CPUs suportadas
 - `arch/arm/mm/proc- $\$$ CPUTYPE.S`
 - `include/asm-arm/procinfo.h`
 - `arch/arm/kernel/setup.c`
 - Tabela contém máscara e valor esperado
 - Se ID recebido do bootloader não confere, morre!
 - `arch/arm/boot/compressed/head.S`

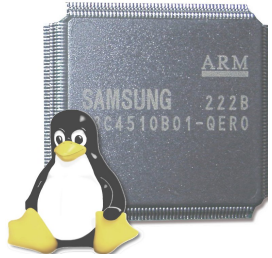


Portando o Linux para uma máquina ARM: kernel

- Segundo passo: **descompressão do kernel**

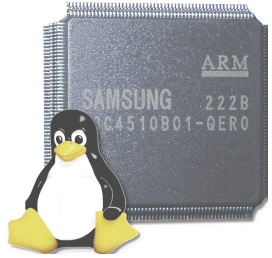
```
decompress_kernel() {  
    arch_decomp_setup();  
    makecrc();  
   _putstr("Uncompressing Linux...");  
    gunzip();  
   _putstr(" done, booting the kernel.\n");  
}
```

- **arch/arm/boot/compressed/misc.c**



Portando o Linux para uma máquina ARM: kernel

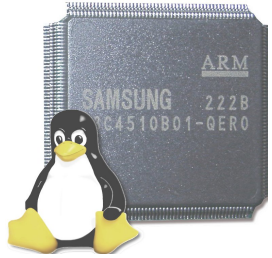
- `arch_decomp_setup()`
 - configuração de watchdog
 - mensagens de checkpoint pela porta serial
 - permite saber se o bootloader cumpriu seu papel
 - `printascii()`, `printch()`
- `arch/arm/kernel/debug.S`



Portando o Linux para uma máquina ARM: kernel

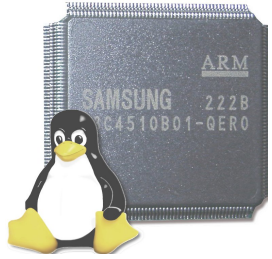
Terceiro passo: [inicialização do kernel](#)

- `start_kernel()` em `linux/init/main.c`
`setup_arch(&cmdline);`
- `setup_arch()` em `arch/arm/kernel/setup.c`
`setup_processor();`
`mdesc = setup_machine();`
`cpu_init();`
`init_arch_irq = mdesc->init_irq;`
`init_machine = mdesc->init_machine;`
`system_timer = mdesc->timer;`



Portando o Linux para uma máquina ARM: kernel

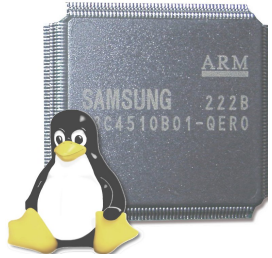
- `setup_machine()`: obtém a *estrutura de descrição da máquina*, que contém:
 - Endereços de RAM, Vídeo e I/O
 - Ponteiros para:
 - Endereço da lista de parâmetros do bootloader
 - Funções de inicialização da máquina e IRQ
 - Funções de timer
 - `init()`, `gettimeoffset()`, `timer_setup()`



Portando o Linux para uma máquina ARM: kernel

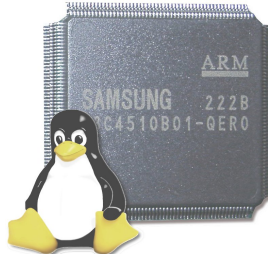
- Quarto passo: **descrevendo a máquina**
 - Inclusão e adaptação de arquivos já existentes
- **arch/arm/mach-s3c2400/mach-gp32.c:**

```
MACHINE_START(GP32, "GamePark GP32")  
    .phys_io      = S3C2400_PA_UART,  
    .boot_params  = S3C2400_SDRAM_PA + 0x100,  
    .map_io       = gp32_map_io,  
    .init_irq     = s3c24xx_init_irq,  
    .timer        = &s3c24xx_timer,  
MACHINE_END
```



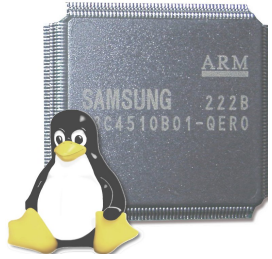
Portando o Linux para uma máquina ARM: kernel

- Headers: `include/asm-arm/arch-XXX/`
- Caso porte envolva a adição de uma nova CPU:
 - `cpu_x.h`
 - `dma.h`
 - `irqs.h`
 - `regs-clock.h`
 - `regs-gpio.h`
 - `regs-mem.h`
 - `regs-serial.h`



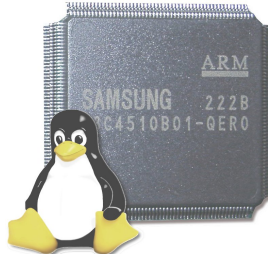
Portando o Linux para uma máquina ARM: kernel

- Rotinas: `arch/arm/mach-xxx/`
- Caso porte envolva a adição de uma nova CPU:
 - `cpu.c` - adição de estrutura da nova CPU
 - `cpu_x.c` - novo arquivo
- `devs.c` - endereços de recursos (USB, LCD, ACD, ...)
- `clock.c` - tabela de clocks da plataforma
- `dma.c` - `dma_irq()`, `dma_config()`, `init_dma()`
- `irq.c` - `ack`, `mask`, `unmask`, `init`



Portando o Linux para uma máquina ARM: kernel

- Detalhes finais
- `arch/arm/mach-XXX`:
 - Makefile - Adição de objetos novos
 - Kconfig - Nova entrada no menu de configuração
- `arch/arm/configs/novo_defconfig`
 - `.config` padronizado para nova plataforma



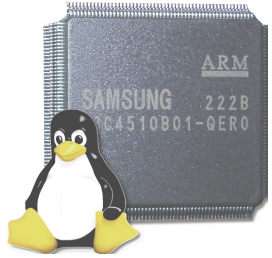
Portando o Linux para uma máquina ARM: drivers

- Device drivers
 - Ficam de fora do arch/arm
 - linux/drivers/subsistema
 - Não devem ser dependentes da arquitetura
 - Sem código asm!

```
#include <arch/lcd_controller.h>
```

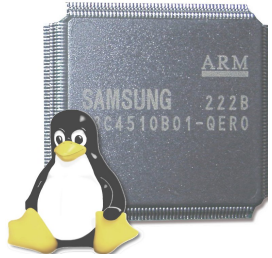
```
arch -> arch-s3c2410/
```

```
arch -> arch-integrator/
```



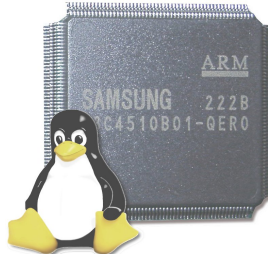
Depuração do kernel

- Putz, não funciona!
 - Debug via serial
 - `printascii()`, `putc()`, `putstr()`
 - Debug via JTAG
 - Acesso ao estado do hardware (ex: registradores)
 - Osciloscópio
 - Mailing lists :-)



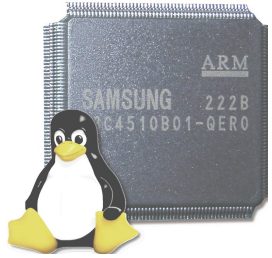
Considerações Finais

- Planeje incorporar patches no kernel mainstream
- Realizar o porte em etapas
 - Base system (clock, UART, IRQ, GPIO, CPU)
 - Extended system (USB, LCD, power managem.)
- Trabalhe com a comunidade
- Desenvolvedores estão escassos, unam-se!
 - Garantia de diversão, ou o seu dinheiro de volta!



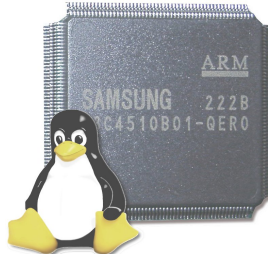
Timers

- Como o kernel mede o tempo?
 - Programando o relógio do sistema (hardware) para gerar interrupções em uma determinada frequência (8254, HPET, RTC, ...)
 - Consultando periodicamente o contador de ciclos (TSC) do processador
- A frequência que será programada no relógio é definida em `<asm/param.h>`, na macro **HZ**
- O valor é dependente de arquitetura, e em geral restrito a uma determinada faixa de valores



Timers

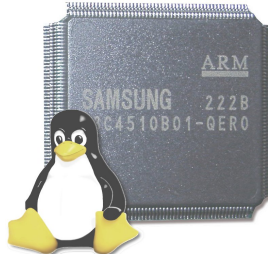
- A variável **jiffies** mantém a quantidade de tiques do relógio até o momento, e será incrementada em uma unidade a uma taxa de HZ vezes por segundo
- Influências de HZ no sistema:
 - Valores considerados baixos (100) resultam em um número menor de interrupções, mas pelo preço de respostas mais lentas do sistema (centesimal)
 - Valores altos (1.000) dão ao sistema respostas rápidas (millesimal) e timers mais precisos, mas causam 10x mais interrupções



BogoMIPS

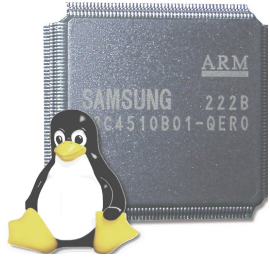
- Contração de “**Bogus**” (falso, vazio, inútil) com **MIPS** (Millions of Instructions Per Second)
- O valor é calibrado pelo kernel a cada inicialização:

```
Calibrating delay using timer specific routine..  
1702.97 BogoMIPS (lpj=3405954)
```
- A função genérica responsável, `calibrate_delay`, encontra-se no arquivo `init/calibrate.c`
- Algumas arquiteturas implementam uma função específica (`frv` e `xtensa`)



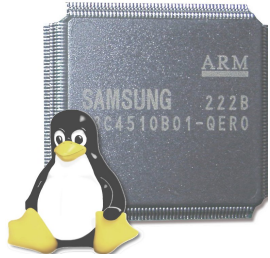
BogoMIPS

- Os BogoMIPS são utilizados em geral por drivers que necessitam de intervalos curtos e precisos de tempo, em loops de espera ocupada (*busy wait*)
- Por causa disso, seu uso deve se limitar a casos onde não haja outra alternativa, pois nesse período de espera o processador não executará trabalho útil
 - `mdelay`(unsigned long milisegundos)
 - `udelay`(unsigned long microsegundos)
 - `ndelay`(unsigned long nanosegundos)



Referências

- Robert Love. *Linux Kernel Development*. 2ª edição. Editora Novell Press, 2005.
- Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman. *Linux Device Drivers*. 3ª edição. Editora O'Reilly, 2005.
- Daniel P. Bovet, Marco Cesati. *Understanding the Linux Kernel*. 3ª edição. Editora O'Reilly, 2005.



Do relógio de pulso a clusters NUMA: Portando o Linux para novas arquiteturas

Felipe W. Damasio

Lucas C. Villa Real

Ricardo Nabinger Sanchez

7º Fórum Internacional de Software Livre