LINUX EN SISTEMAS EMPOTRADOS

Segunda Parte

¿Qué necesitamos saber sobre Linux para empotrarlo?

UAM 26 Abril 2006

LINUX EN SISTEMAS EMPOTRADOS

¿Qué necesitamos saber sobre Linux?

- **Diseño del núcleo del sistema**
- **Programación de drivers**
- **Herramientas del sistema**
- **Sistemas de almacenamiento**

LINUX EN SISTEMAS EMPOTRADOS

¿Qué necesitamos saber sobre Linux?

- **Diseño del núcleo del sistema**
- **Programación de drivers**
- **Herramientas del sistema**
- **Sistemas de almacenamiento**

¿Qué necesitamos saber sobre el núcleo de Linux?

- Diseño basado en UNIX
- Monolítico
- Modular
- Multitarea (Multiprocesador)
- ¿"*Preemptivo*"?

¿Qué necesitamos saber sobre el núcleo de Linux?

- **Diseño basado en UNIX**
  - Dos modos de ejecución: usuario y supervisor
  - Protección basada en MMU(hardware)
  - uCLinux: dos modos de ejecución por SW. No hay protección.
  - Servicios del núcleo: llamadas al sistema (POSIX)

¿Qué necesitamos saber sobre el núcleo de Linux?

- **Los módulos del kernel**
  - ➔Funcionalidades sólo cuando se necesiten
  - ➔Permiten maximizar el uso de la memoria
  - ➔Facilitan el desarrollo de drivers.

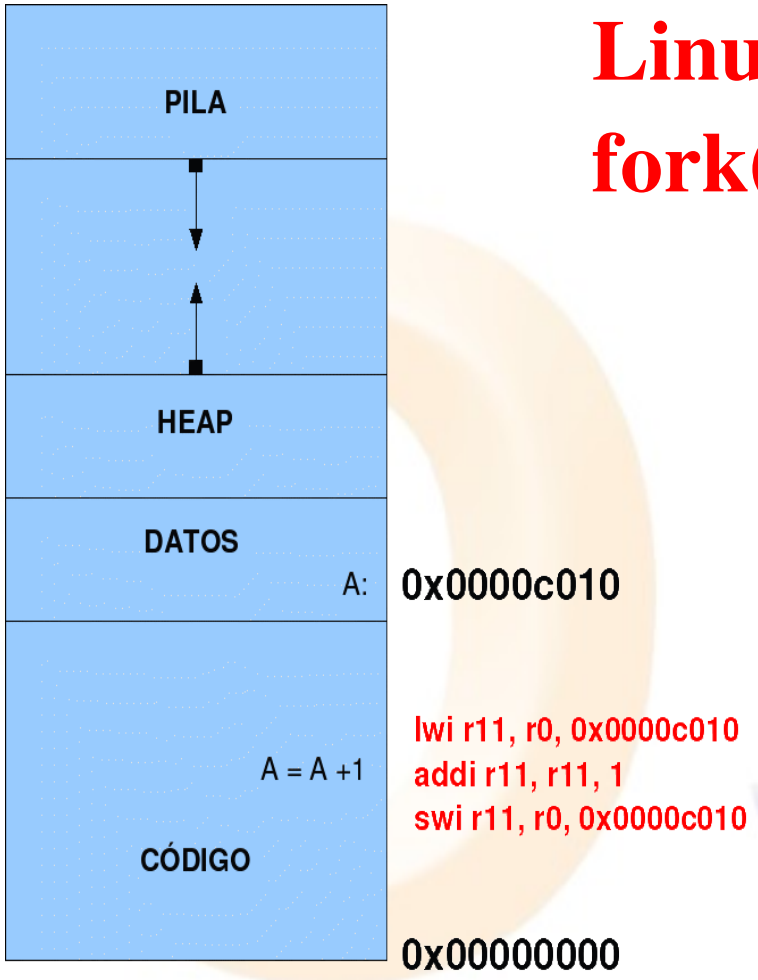
¿Qué necesitamos saber sobre el núcleo de Linux?

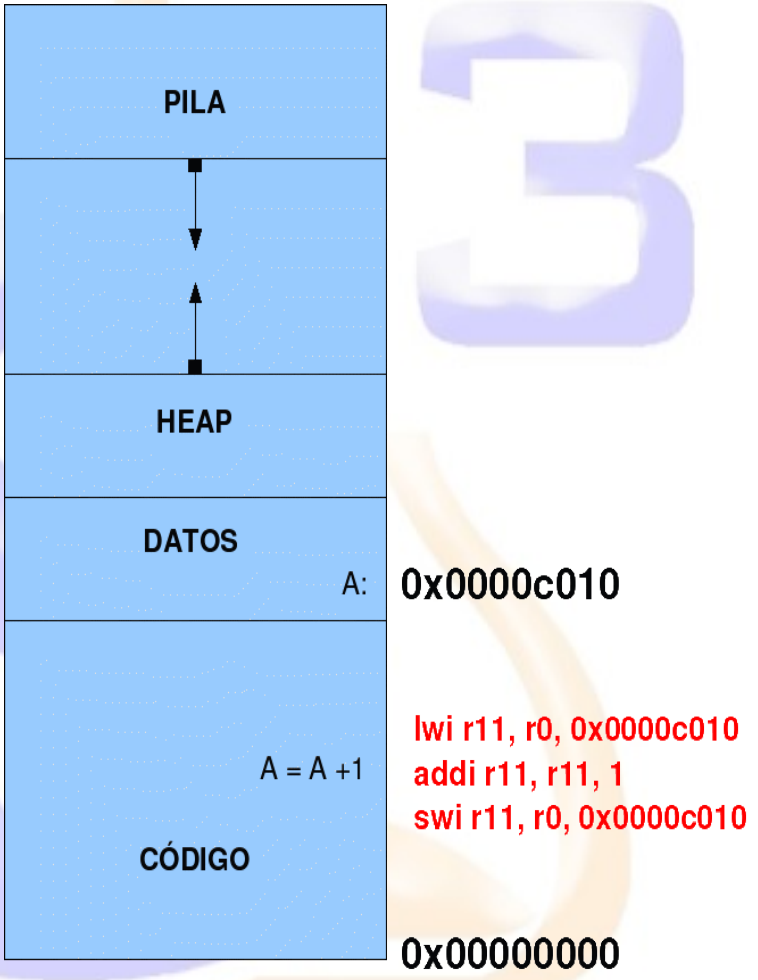
- **Multitarea**
  - Planificación expulsiva basada en prioridades
  - Planificación de “tiempo real”
  - Linux 2.4: No preemptive
  - Linux 2.6: Full preemptive (mejora los tiempos de respuesta)

¿Qué necesitamos saber sobre el núcleo de Linux?

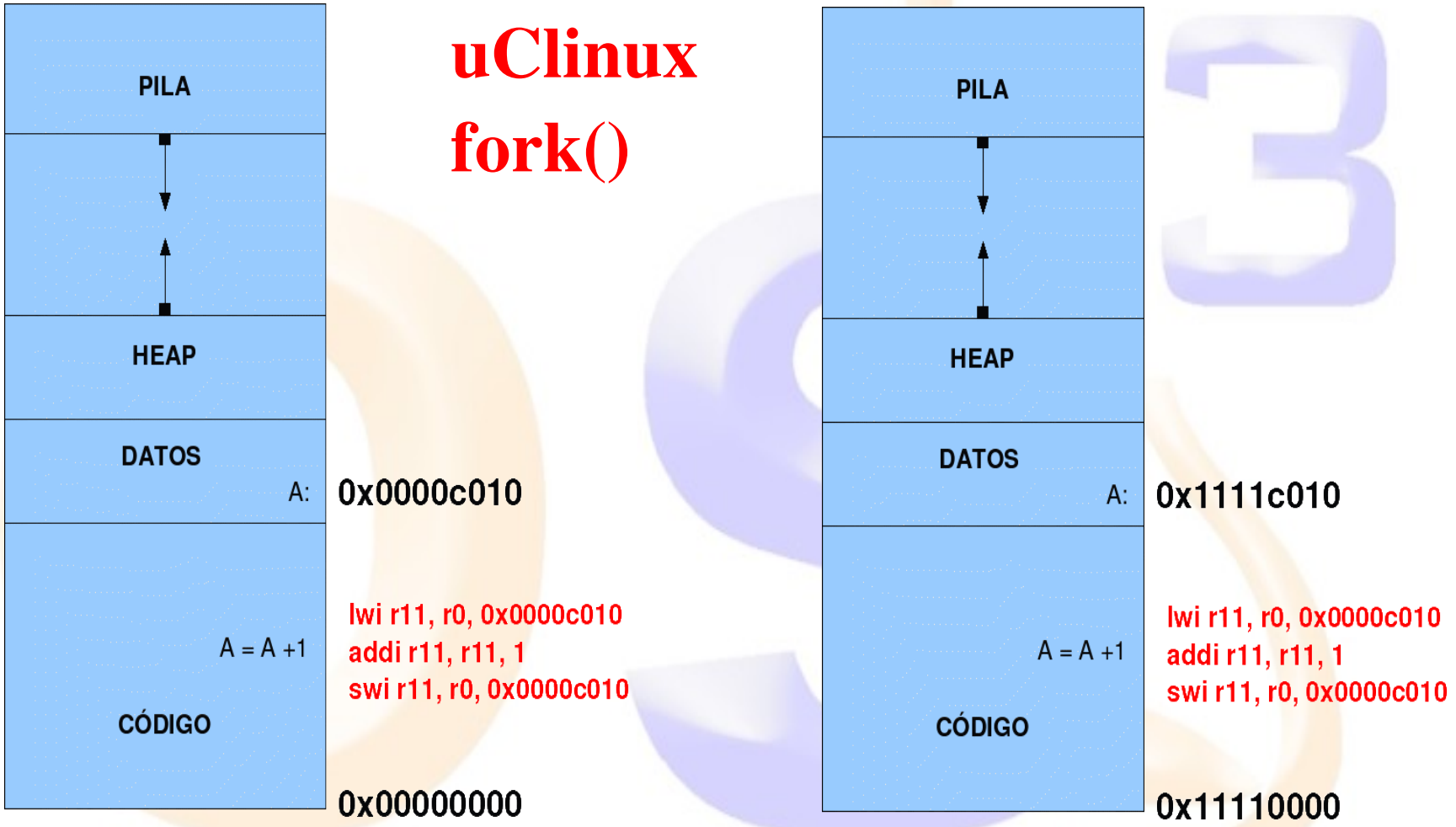
- **uClinux**
  - Creación de procesos con vfork()
  - No existe copy on write
  - memoria dinámica sin heap (brk, sbrk)
  - No hay swap
  - Pila de usuario estática
  - No existe mmap

Linux fork()





Direcciones virtuales!



Direcciones físicas!

LINUX EN SISTEMAS EMPOTRADOS

¿Qué necesitamos saber sobre Linux?

- **Diseño del núcleo del sistema**
- **Programación de drivers**
- **Herramientas del sistema**
- **Sistemas de almacenamiento**

Programación de drivers

- **API:**
  - open, close, read, write, ioctl, mmap
  - petición de interrupciones
  - petición de memoria
  - wait_queues: espera de eventos
  - timers, esperas activas

Programación de drivers

- **Tratamiento de interrupciones:**
  - Se fomenta el anidamiento (máxima eficiencia)
  - No hay un proceso especializado
  - División entre top_half y bottom_half
    - top_half: parte crítica (sin esperas)
    - bottom_half: Posible ejecución con retraso. Interrupciones habilitadas

Programación de drivers

- **Tratamiento de interrupciones:**
  - **Bottom_half con:**
    - softirqs (SMP, estáticas)
    - tasklets (SMP*, dinámicas)
    - bottom halves (estáticas)

Programación de drivers

- **Gestión de la memoria**
  - kmalloc (kfree): prioridad y tipo
  - lookaside_caches: eficiencia vs consumo de recursos

Programación de drivers

- **Timers**
  - Resolución:
    - Linux 2.4: 10 milisegundos
    - Linux 2.6: 1 milisegundo
  - Esperas activas: udelay() y mdelay()
  - High Resolution timers (HPET)

Programación de drivers

- **/proc**
  - sistema de ficheros virtual
  - permite acceder a datos internos del kernel
  - Muy útil para depuración
  - Puede sustituir a ioctl

Programación de drivers

- **kernel threads y task queues**
  - Ejecución de código “asíncrono” respecto al sistema
  - Utilizadas por Linux para diversas tareas
  - Linux empotrado 

LINUX EN SISTEMAS EMPOTRADOS

¿Qué necesitamos saber sobre Linux?

- **Diseño del núcleo del sistema**
- **Programación de drivers**
- **Herramientas del sistema**
- **Sistemas de almacenamiento**

Sistema GNU Linux

- **Linux OS (kernel ó núcleo)**

+

- **Software GNU: herramientas básicas, librerías, programas**

Herramientas del sistema: ejecución

- El shell UNIX: intérprete de comandos
- Comandos básicos: ls, ps, cat, grep, awk ...
- Programación de shell scripts
- Limitaciones en uClinux

Herramientas del sistema: desarrollo

- gcc, as, ld, make, ar, objdump, strip, gdb
- Compilación cruzada:
  - Host (i386) y target (ARM, PowerPC, Microblaze, ...)
  - **Toolchain**: binutils + libc
  - Bien soportado en la mayoría de arquitecturas

Herramientas del sistema: depuración

- Depuración: GDB, Linux Trace Toolkit, KDBG, Novell DBG, Kprobes, ...
- bgcc, gprof, gcov, ...
- Algunas herramientas permiten depuración remota: GDB, LTT

LINUX EN SISTEMAS EMPOTRADOS

¿Qué necesitamos saber sobre Linux?

- **Diseño del núcleo del sistema**
- **Programación de drivers**
- **Herramientas del sistema**
- **Sistemas de almacenamiento**

Sistemas de almacenamiento

- **Discos magnéticos:** bien soportados por Linux (ATA, SATA, SCSI)
- **Linux & RAID's** (Mirroring, stripping)
- **Almacenamiento remoto (NFS):** BOOTP
- **Memoria Flash**

Sistemas de almacenamiento

- **Memoria FLASH:**
  - NAND vs NOR
  - lecturas, borrado-escrituras
  - Tiempo de vida limitado
  - Necesario wear-leveling

Sistemas de almacenamiento

Memoria FLASH: NAND vs NOR

	NOR	NAND
capacidad	baja	alta
lecturas	rápidas	rápidas
escrituras	lentas (seg)	rápidas(ms)
acceso	aleatorio	secuencial
XIP	si	no

Sistemas de almacenamiento

- **Memoria FLASH: Fiabilidad**
  - Bit-Flipping: Algoritmo EDC/ECC
  - Bad Block handling (NAND)
  - Wear-leveling

Sistemas de almacenamiento

- **Memoria FLASH: Soporte Linux**
  - Interface MTD's en Linux:
    - Raw char, Raw block
    - FTL, NFTL
  - JFFSx, YAFFS
  - Compact Flash (Flash con Interface IDE)
  - DiskOnChip (Msystems, TrueFFS)

LINUX EN SISTEMAS EMPOTRADOS

Creación de un sistema empotrado basado en Linux

- **Inicialización**
- **El sistema de ficheros**
- **Librerías y aplicaciones**
- **Optimización de recursos**

Creación de un sistema empotrado

Inicialización:

- Elección y configuración del kernel
- Bootloader
- El proceso Init: System V, Busybox

Creación de un sistema empotrado

Inicialización:

- **Elección y configuración del kernel basada en:**
  - Funcionalidades necesarias
  - Prestaciones requeridas
  - Consumo de recursos (RAM)

Creación de un sistema empotrado

Inicialización:

- **Bootloader**
  - Inicializa el HW base*
  - Descomprime y copia el kernel en RAM
  - Dependiente de arquitectura
  - Monitores: Das-UBoot

Creación de un sistema empotrado

Inicialización:

- **El proceso Init:**
  - Bootloader copia* el kernel a memoria y lo ejecuta
  - El kernel realiza inicializaciones de interrupciones, MMU, dispositivos, estructuras, arrays, ...
  - Monta el root file system
  - Crea el proceso (kernel thread) init
  - Se ejecuta el programa /sbin/init

Creación de un sistema empotrado

Inicialización:

- **El proceso Init:**
  - Standar System V: niveles de ejecución
  - Busybox
  - Aplicación específica

LINUX EN SISTEMAS EMPOTRADOS

Creación de un sistema empotrado basado en Linux

- **Inicialización**
- **El sistema de ficheros**
- **Librerías y aplicaciones**
- **Optimización de recursos**

Creación de un sistema empotrado

El sistema de ficheros:

- *Soporte físico (Magnéticos, Flash)*
- *Soporte lógico*
- *Estructura*

Creación de un sistema empotrado

El sistema de ficheros: Soporte lógico

- *Hard Disk (+xFTL): Ext2, Ext3, ReiserFS*
- *FLASH (RW): JFFSx, YAFFS, TrueFFS**
- *FLASH (RO): Cramfs, SquashFS, romfs*
- *eXecution In Place?*

Creación de un sistema empotrado

El sistema de ficheros: Soporte lógico

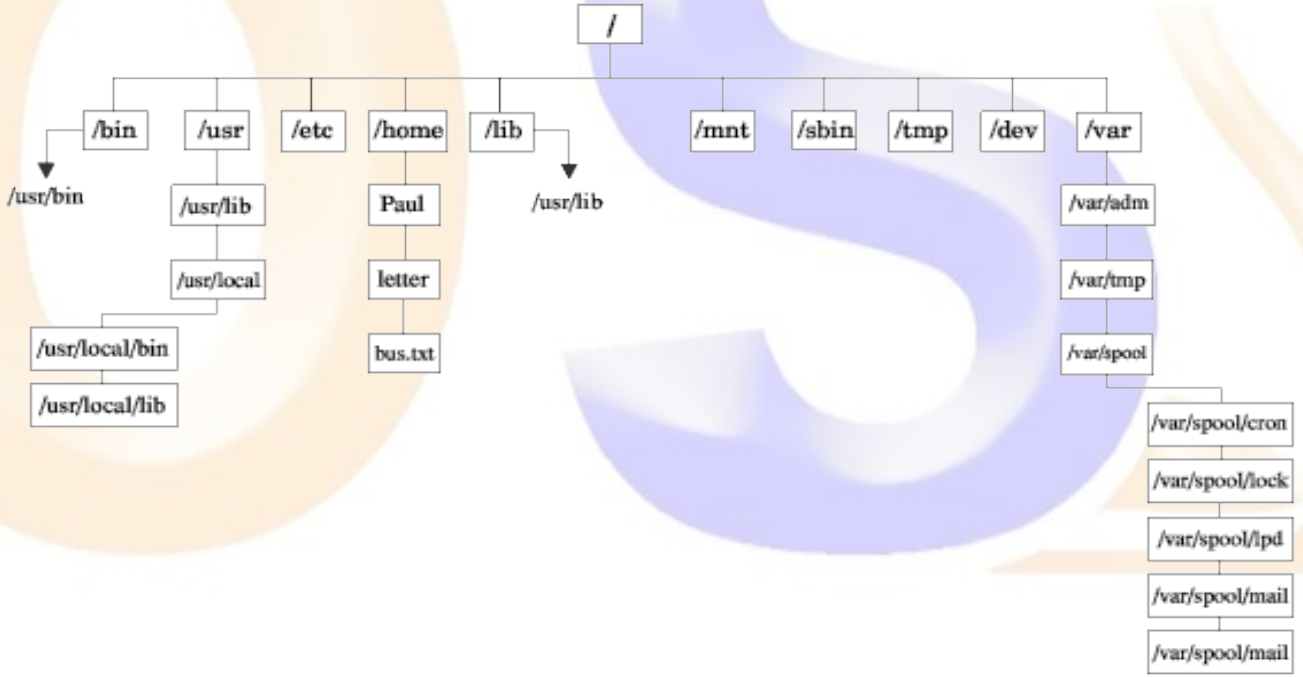
- *NFS (Network File System)*
- *RamFS, TmpFS (root file system, temporal data)*
- *InitRamFS (2.6, máxima eficiencia)*

LINUX EN SISTEMAS EMPOTRADOS

Creación de un sistema empotrado

El sistema de ficheros: Estructura

Unix FHS (File System Hierarchy Standard)



Creación de un sistema empotrado

El sistema de ficheros: Estructura

Unix FHS (File System Hierarchy Standard

- Root file system: /
- Directorios top-level con funcionalidad específica
- Programas GNU (POSIX) esperan encontrar FHS

LINUX EN SISTEMAS EMPOTRADOS

Creación de un sistema empotrado basado en Linux

- **Inicialización**
- **El sistema de ficheros**
- **Librerías y aplicaciones**
- **Optimización de recursos**

UAM 26 Abril 2006

Creación de un sistema empotrado

Qué hay que poner en el sistema de ficheros?

- Programas básicos, específicos: /bin, /sbin
- librerías: /lib, /usr/lib
- ficheros de configuración: /etc
- ficheros especiales de dispositivos: /dev

Creación de un sistema empotrado

Cómo se hace (y I)?

- Identificando ficheros en una distribución estándar (ldd, strace, rpm)
- Copiar a la estructura creada
- Probar: chroot ó NFS
- Ideal cuando Target = Host

Creación de un sistema empotrado

Cómo se hace (y II)?

- **Linux From Scratch:**
  - Proyecto recopilación código fuente de GNU Linux: programas, librerías, ficheros de configuración, información de dependencias
  - Se debe compilar lo que necesitemos
  - Necesario cuando Target != Host

Creación de un sistema empotrado

Cómo se hace (y III)?

- **BusyBox**
  - Pensado para Linux empotrado
  - Combina pequeñas versiones de los comandos básicos en un único ejecutable.
  - optimiza el tamaño utilizado (text shared)

Creación de un sistema empotrado

Librerías

- librería básica del sistema: **libc**
  - glibc: 1.7MB
  - uClibc: 400Kbytes (buildroot)
- librerías dinámicas vs estáticas

Creación de un sistema empotrado

Librerías estáticas

- Dependencias resueltas en tiempo de compilación
- Binarios autónomos
- Código replicado (en memoria y en disco/flash)

Librerías dinámicas

- Dependencias resueltas en tiempo de ejecución
- Necesidad de ld.so + librerías en el sistema
- Código compartido

Creación de un sistema empotrado

- uClinux no soporta librerías dinámicas directamente ya que se necesita MMU para su implementación.

- Algunas arquitecturas (Coldfire, ARM7) soportan librerías dinámicas con algunas limitaciones

LINUX EN SISTEMAS EMPOTRADOS

Creación de un sistema empotrado basado en Linux

- **Inicialización**
- **El sistema de ficheros**
- **Librerías y aplicaciones**
- **Optimización de recursos**

UAM 26 Abril 2006

Optimización de recursos

- **ARM:** THUMB, bits para loops y rotación
- **XIP:** eXecution in Place (NOR Flash)
- **Busybox:** shared text section
- **Librerías dinámicas vs estáticas**

Optimización de recursos

¿Como reducir el espacio utilizado en memoria y en FLASH?

- Elección adecuada del kernel
- Explotar la modularidad de Linux
- Opciones de optimización en la compilación de los ejecutables y del Kernel
- XIP: código del kernel = 1MB

Optimización de recursos

¿Como reducir el espacio utilizado en memoria y en FLASH?

- JFFS2, CRAMFS: compresión de código hasta ejecución
- uClibc
- Stripped executables
- Reducir el tamaño de las librerías (Library Optimizer Tool)
- Kernel tuning: buffers, caches

Optimización de recursos

¿Como mejorar el tiempo de arranque?

- **Prelink**: ld.so por adelantado (mejoras del 50%)
- **initramfs**: ejecutables y librerías en cache

Optimización de recursos

Opciones de configuración

- **Kernel 2.6:**
  - 1.2MB (515KB comprimido)
- **Kernel 2.6: CONFIG_EMBEDDED**
  - 766KB (323KB comprimido)

Conclusiones

- Linux paradigma del Open Source
- Preparado para sistemas empotrados
- Portado a una gran gama de procesadores
- En sistemas de control y/o adquisición de datos: Linux + extensión de tiempo real
- Mejoras futuras: soporte técnico, tiempo de inicialización, control de energía

LINUX EN SISTEMAS EMPOTRADOS

Alejandro Lucero (OS3, SL)

alucero@os3sl.com

informacion@os3sl.com

www.os3sl.com

UAM 26 Abril 2006