Introducción al desarrollo de aplicaciones para sistemas Linux empotrados basados en el microprocesador ARM Intel Xscale PXA270

José Manuel Cano García jcgarcia@uma.es

Departamento de Tecnología Electrónica. Universidad de Málaga.

Laboratorio de Diseño de Sistemas Digitales

Febrero de 2007

Índice

1. Introducción

En este documento se recogen algunas instrucciones para empezar a desarrollar aplicaciones para sistemas empotrados basados en el Intel XScale PXA270 bajo el sistema operativo Linux, utilizando para ello la plataforma de desarrollo disponible en el laboratorio basada en los módulos Colibrí de Toradex.

Para el desarrollo de estas aplicaciones se cuenta con un PC de trabajo con sistema operativo Linux, y que dispone de las herramientas de desarrollo necesarias para realizar la compilación cruzada de aplicaciones que posteriormente se podrán ejecutar en el sistema empotrado. El sistema empotrado también dispone de sistema operativo Linux, sobre el que se ejecutarán las aplicaciones desarrolladas.

Este documento recoge cómo configurar y conectar la plataforma empotrada al PC de desarrollo, así como los pasos necesarios para compilar y probar las aplicaciones desarrolladas. En ningún caso este documento es una introducción al sistema operativo Linux. Información introductoria y avanzada sobre este sistema operativo puede consultarse en la página web *The Linux Documentation Project* [1].

2. El hardware del sistema empotrado

Los módulos Colibrí de Toradex cuentan con un microprocesador Intel ARM XScale PXA270, similar al que incorporan algunas PDAs. Además integran 64 MB de SDRAM, 32 MB de FLASH, y un controlador de Ethernet (ver fig 1). Los módulos se encuentran conectados a una placa de expansión Orchid, que incorpora los conectores y la lógica necesaria para los múltiples periféricos que integra el PXA270, contando con conectores para puertos serie RS232, conectores USB, Conector Ethernet RJ45, conectores de audio, zócalo para tarjetas compact-flash, zócalo para memoria SD-MMC, puerto IrDA, etc. (ver fig 2).

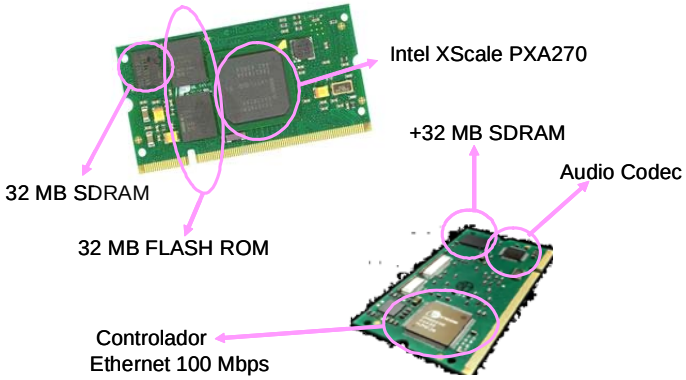

Figura 1: El módulo colibri PXA 270

Para facilitar el desarrollo de aplicaciones, el sistema empotrado se conectará al PC de trabajo mediante un cable Ethernet cruzado que permite comunicar ambos equipos en red. Además, uno de los puertos serie del sistema empotrado (el conector inferior) puede conectarse al PC de desarrollo o a otro PC auxiliar de forma que actúe como consola de terminal del sistema empotrado. Para ello será necesario la ejecución de una aplicación de consola de terminal serie como por ejemplo Hyperterminal de Windows en el PC auxiliar. La configuración del puerto

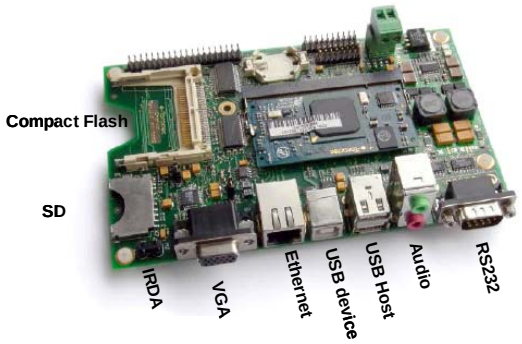

Figura 2: La placa de desarrollo Orchid

serie para este propósito es de 9600 bps, sin paridad y sin control de flujo. El conexionado se esquematiza en la figura 3.

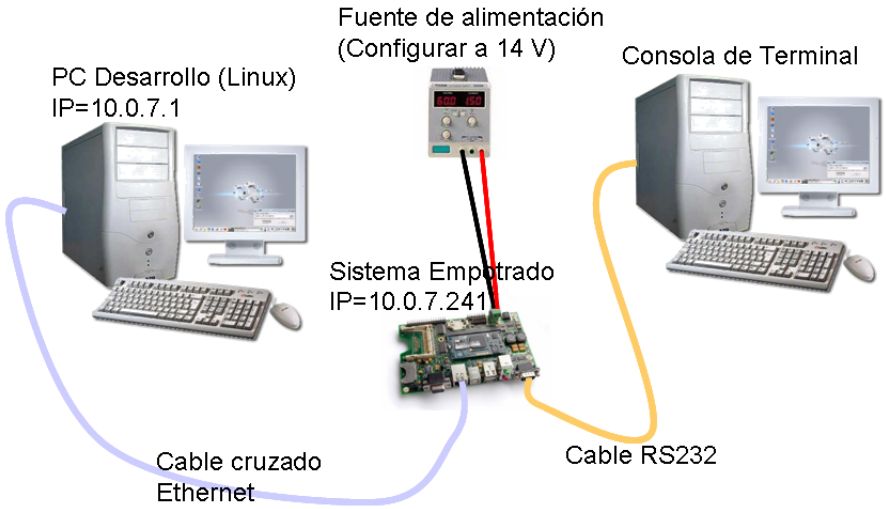

Figura 3: Esquema de conexionado del sistema empotrado

La placa del sistema empotrado debe ser alimentada suministrando una tensión mediante una fuente de alimentación externa. En el laboratorio se dispone de cables que integran un regulador lineal de 12 V, que generan una tensión adecuada para el funcionamiento de dicho módulos. Para el correcto funcionamiento del sistema, se debe programar la fuente de alimentación para dar una tensión ligeramente superior (13.5-14 V). Se ruega encarecidamente no alimentar el sistema a tensiones mayores, pues aunque el regulador protege al módulo de sobretensiones, una mayor caída de tensión en el regulador hace que éste disipe mayor potencia y pueda llegar a quemarse.

3. Componentes del software del sistema empotrado

El software de un sistema empotrado basado en Linux tiene esencialmente 3 bloques bien diferenciados: El gestor de arranque o *bootloader*, el *kernel* y el sistema de ficheros raíz (*root filesystem*). El *bootloader* es una aplicación básica que inicializa el microcontrolador tras el encendido del mismo y es capaz de hacer algunas operaciones sencillas destinadas principalmente a la carga del *kernel* y arranque del sistema operativo. El *kernel* o núcleo es el principal elemento

del sistema operativo, ya que es el que se encarga de la planificación de tareas y del control del hardware y de la memoria, ofreciendo un marco de ejecución genérico a las aplicaciones de usuario. El sistema de ficheros raíz es un árbol de directorios que contiene ficheros de configuración, aplicaciones y librerías dinámicas, y en el caso de un sistema linux tiene una estructura más o menos genérica. Para más información sobre los componentes de un sistema Linux empotrado puede consultarse el libro *Building Embedded Linux System* [2].

3.1. Bootloader

En el caso del sistema que nos ocupa, el bootloader es el u-boot y se encuentra grabado en el primer sector de la memoria flash de los módulos colibrí, de manera que se ejecuta inmediatamante tras el encendido del sistema. El U-boot proporciona un acceso básico a la flash, permitiendo borrar y escribir sectores, y copiar información de la RAM a la flash o al revés. Además, también es capaz de realizar algunas operaciones básicas de red utilizando el controlador de ethernet, permitiendo obtener una dirección ethernet por *dhcp* y descargar ficheros de un servidor preconfigurado mediante el protocolo *tftp*. Estas características se utilizarán, como más tarde se comenta en la sección 4 para descargar el ejecutable del kernel desde el PC de desarrollo.

3.2. Kernel

Existen varias versiones de kernel de Linux, con diferentes características. Para el sistema empotrado se utilizarán 2 versiones diferentes, la versión 2.4.29 y la versión 2.6.12, que se han compilado y se encuentran disponibles en el PC de desarrollo. Más adelante en este se comentará cómo se realiza la compilación del kernel. Los ejecutables de estas versiones del kernel se encuentran en el directorio `/tftproot` para que puedan ser descargados por el u-boot durante el arranque del sistema empotrado. La versión 2.4.29 soporta mayor número de periféricos, incluido el controlador de VGA, por lo que permite la ejecución de aplicaciones gráficas. Por otra parte, el kernel 2.6 tiene una estructura más moderna y es el que se utilizará para el desarrollo de drivers que se comenta más adelante en este documento.

3.3. Sistema de ficheros raíz

El sistema de ficheros que se va a utilizar ha sido creado con la herramienta OpenEmbedded [3], y se encuentra almacenado en el directorio `/colibri` del PC de desarrollo. Dicho directorio es exportado mediante el sistema NFS de manera que puede ser compartido en red por otros equipos. El sistema empotrado se ha configurado de forma que tras el arranque del kernel monta como sistema de ficheros raiz el directorio compartido, para lo cual es necesario mantenerlo conectado al PC de desarrollo mediante el cable de red Ethernet.

4. Arranque del sistema

Para configurar el puesto de trabajo hay que conectar el sistema empotrado al PC de desarrollo mediante el cable Ethernet, y al PC que se vaya a utilizar como consola mediante el cable RS-232. Puesto que el PC de desarrollo va a servir la imagen del kernel y el sistema de ficheros raíz al sistema empotrado, es necesiario haber completado previamente el arranque del sistema

operativo Linux en el PC de desarrollo, de manera que queden activados los servicios de *dhcp*, *tftp* y *nfs*

4.1. Procedimiento de arranque por defecto

Tras el encendido de la fuente de alimentación que suministra tensión al sistema empotrado, lo primero que se ejecuta automáticamente es el gestor de arranque U-boot. El gestor de arranque inicializa el procesador y transcurridos unos segundos sin que se produzca la pulsación de una tecla por el usuario, intentará arrancar el sistema con las opciones por defecto. En el arranque por defecto el u-boot obtiene automáticamente del PC de desarrollo una dirección IP para el sistema empotrado mediante el protocolo *dhcp*, y una vez realizado esto, procede a descargar la imagen del kernel desde el PC de desarrollo utilizando el protocolo *tfpt*. La imagen del kernel que se utiliza por defecto es la de la versión 2.6.12, que está almacenada en el fichero `/tftproot/uImage26` del PC de desarrollo. Una vez descargada la imagen del kernel, la descomprime y ejecuta, pasando por tanto el control al sistema operativo Linux. Una vez arrancado el kernel, éste configura los diferentes periféricos y protocolos e intenta acceder al sistema de ficheros raiz para comenzar a ejecutar las aplicaciones básicas de configuración y control. La configuración por defecto del kernel hace que intente montar como directorio raíz el sistema de ficheros en red exportado por el PC de desarrollo mediante el protocolo *nfs*. Si este montaje se realiza adecuadamente, continúa el arranque del sistema poniendo en marcha diferentes servicios, hasta que finalmente arranca la consola de acceso al sistema. A continuación se incluye un volcado de la información que va dando el sistema en el proceso de arranque por la consola serie:

```
U-Boot 1.1.2 (Nov 22 2006 - 17:20:43)

U-Boot code: A3F80000 -> A3F98B34  BSS: -> A3F9D27C
RAM Configuration:
Bank #0: a0000000  0 kB
Bank #1: 00000000  0 kB
Bank #2: 00000000  0 kB
Bank #3: 00000000  0 kB
Flash:  0 kB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Error: start and/or end address not on sector boundary
CPU speed: 312000kHz
Hit any key to stop autoboot:  5  4  3  2  1  0
## Booting image at 00080000 ...
Bad Magic Number
dm9000 i/o: 0x08000000, id: 0x90000a46
MAC: 00:14:2d:00:09:c0
operating at 100M full duplex mode
BOOTP broadcast 1
BOOTP broadcast 2
DHCP client bound to address 10.0.7.241
TFTP from server 10.0.7.1; our IP address is 10.0.7.241
Filename '/tftproot/uImage26'.
Load address: 0xa1000000
Loading: *#################################################################
 ################################################################
 ################################################################
 ##########################################
done
Bytes transferred = 1223837 (12ac9d hex)
## Booting image at a1000000 ...
   Image Name:   Linux Kernel Image
   Created:      2007-01-23  15:20:21 UTC
   Image Type:   ARM Linux Kernel Image (gzip compressed)
```

```
   Data Size:    1223773 Bytes =  1.2 MB
   Load Address: a0008000
   Entry Point:  a0008000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK

Starting kernel ...

Linux version 2.6.12.4-col2 (root@pc7131te) (gcc version 3.4.4) #1 Tue Jan 23 16:17:24 CET 2007
CPU: XScale-PXA270 [69054117] revision 7 (ARMv5TE)
CPU0: D VIVT undefined 5 cache
CPU0: I cache: 32768 bytes, associativity 32, 32 byte lines, 32 sets
CPU0: D cache: 32768 bytes, associativity 32, 32 byte lines, 32 sets
Machine: Toradex Colibri Module
Ignoring unrecognised tag 0x00000000
Ignoring unrecognised tag 0x00000000
Ignoring unrecognised tag 0x00000000
Ignoring unrecognised tag 0x00000000
Memory policy: ECC disabled, Data cache writeback
Run Mode clock: 208.00MHz (*16)
Turbo Mode clock: 312.00MHz (*1.5, active)
Memory clock: 208.00MHz (/2)
System bus clock: 208.00MHz
Built 1 zonelists
Kernel command line: root=/dev/nfs ip=:::::eth0: console=ttyS0,9600n8
PID hash table entries: 512 (order: 9, 8192 bytes)
Console: colour dummy device 80x30
Dentry cache hash table entries: 16384 (order: 4, 65536 bytes)
Inode-cache hash table entries: 8192 (order: 3, 32768 bytes)
Memory: 64MB = 64MB total
Memory: 62208KB available (2133K code, 403K data, 104K init)
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
NET: Registered protocol family 16
SCSI subsystem initialized
Linux Kernel Card Services
  options:  [pm]
usbcore: registered new driver hub
TC classifier action (bugs to netdev@vger.kernel.org cc hadi@cyberus.ca)
NetWinder Floating Point Emulator V0.97 (double precision)
JFFS2 version 2.2. (C) 2001-2003 Red Hat, Inc.
Initializing Cryptographic API
ttyS0 at MMIO 0x40100000 (irq = 22) is a FFUART
ttyS1 at MMIO 0x40200000 (irq = 21) is a BTUART
ttyS2 at MMIO 0x40700000 (irq = 20) is a STUART
io scheduler noop registered
io scheduler anticipatory registered
io scheduler deadline registered
io scheduler cfq registered
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
loop: loaded (max 8 devices)
dm9000 Ethernet Driver
eth0: dm9000 at f4000000,f4000004 IRQ 146 MAC: 00:14:2d:00:09:c0
Uniform Multi-Platform E-IDE driver Revision: 7.00alpha2
ide: Assuming 50MHz system bus speed for PIO modes; override with idebus=xx
Probing Colibri flash at physical address 0x00000000 (32-bit buswidth)
Colibri flash: Found 2 x16 devices at 0x0 in 32-bit bank
 Intel/Sharp Extended Query Table at 0x010A
  Unknown Intel/Sharp Extended Query version 1.4.
gen_probe: No supported Vendor Command Set found
Colibri PCMCIA
usbmon: debugs is not available
Setting port 3 power failed.
pxa27x-ohci pxa27x-ohci: PXA27x OHCI
pxa27x-ohci pxa27x-ohci: new USB bus registered, assigned bus number 1
pxa27x-ohci pxa27x-ohci: irq 3, io mem 0x4c000000
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
Initializing USB Mass Storage driver...
usbcore: registered new driver usb-storage
USB Mass Storage support registered.
usbcore: registered new driver usbhid
drivers/usb/input/hid-core.c: v2.01:USB HID core driver
```

```
mice: PS/2 mouse device common for all mice
Colibri MMC/SD setup done.
NET: Registered protocol family 2
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP established hash table entries: 4096 (order: 3, 32768 bytes)
TCP bind hash table entries: 4096 (order: 2, 16384 bytes)
TCP: Hash tables configured (established 4096 bind 4096)
NET: Registered protocol family 1
eth0: link down
Sending BOOTP requests ..<6>eth0: link up, 100Mbps, full-duplex, lpa 0x41E1
.. OK
IP-Config: Got BOOTP answer from 10.0.7.1, my address is 10.0.7.241
IP-Config: Complete:
      device=eth0, addr=10.0.7.241, mask=255.255.255.0, gw=10.0.7.254,
     host=colibri, domain=dte.uma.es, nis-domain=(none),
     bootserver=10.0.7.1, rootserver=10.0.7.1, rootpath=/colibri
Looking up port of RPC 100003/2 on 10.0.7.1
Looking up port of RPC 100005/1 on 10.0.7.1
VFS: Mounted root (nfs filesystem).
Freeing init memory: 104K

INIT: version 2.86 booting
Loading /etc/keymap-2.6.map
Starting the hotplug events dispatcher udevd
Synthesizing the initial hotplug events
modprobe: module mousedev not found.
modprobe: failed to load module mousedev
modprobe: module evdev not found.
modprobe: failed to load module evdev
modprobe: module joydev not found.
modprobe: failed to load module joydev
Waiting for /dev to be fully populated
Bluetooth: Core ver 2.7
NET: Registered protocol family 31
Bluetooth: HCI device and connection manager initialized
Bluetooth: HCI socket layer initialized
Bluetooth: L2CAP ver 2.7
Bluetooth: L2CAP socket layer initialized
Bluetooth: HIDP (Human Interface Emulation) ver 1.1
NET: Registered protocol family 10
Disabled Privacy Extensions on device c0249750(lo)
failed to load transform for md5
IPv6 over IPv4 tunneling driver
Bluetooth: RFCOMM ver 1.5
Bluetooth: RFCOMM socket layer initialized
Bluetooth: RFCOMM TTY layer initialized
ln: /etc/resolv.conf: File exists
Setting up IP spoofing protection: rp_filter.
Configuring network interfaces... done.
Starting portmap daemon: portmap.
Nothing to be done

INIT: Entering runlevel: 5
Starting Dropbear SSH server: dropbear.
Starting advanced power management daemon: No APM support in kernel
(failed.)
Starting IrDA: .
Starting PCMCIA services: cardmgr[4771]: watching 1 socket
done.
Starting syslogd/klogd: done
Starting thttpd.
Starting Bluetooth subsystem: hcid sdpd rfcomm.
Starting the OBEX Push daemon: opd.
Starting Opie in 5 seconds... press key to interrupt.
You seem to already have a /home/root/Applications directory.
Assuming it is the Opie Applications directory. Exiting.
Starting Opie....
ODevice() - found 'Hardware : Toradex Colibri Module'
ODevice() - unknown hardware - using default.
OGlobal::creating global configuration instance.
OConfig::OConfig()
<unknown>: ODevice reports transformation to be 0
<unknown>: QWS_DISPLAY already set as 'Transformed:Rot0:0' - overriding ODevice transformation
```

```
qt_init() - starting in daemon mode...

OpenZaurus 3.5.4 c7x0 ttyS0

c7x0 login:
```

Una vez completado el arranque se puede acceder al sistema empotrado para ejecutar aplicaciones mediante la consola de terminal si se suministran un nombre de usuario y una contraseña válidos. La contraseña de la cuenta con privilegios de superusuario (*root*) es *pxa270*.

4.2. Arranque con una versión del kernel diferente

Como ya se ha mencionado, la imagen por defecto del kernel corresponde a la versión 2.6.12, que es un kernel de última generación, aunque para este dispositivo no ofrece soporte para algunos periféricos, como por ejemplo el controlador de VGA. El sistema se puede arrancar también con un kernel más antiguo, versión 2.4.29, que sí ofrece soporte para dicho controlador. En esta sección se recoge el procedimiento para arrancar con el kernel 2.4.

Para arrancar con una configuración diferente a la por defecto es necesario detener el arranque normal del u-boot pulsando una tecla en la consola de terminal cuando se indique. Al pulsar una tecla, u-boot detiene su ejecución y abre una sesión interactiva que permite al usuario ejecutar un conjunto básico de comandos. Para arrancar el kernel 2.4, será necesario en primer lugar obtener una dirección IP para el sistema empotrado, lo que se consigue mediante el comando *dhcp*:

```
u-boot$ dhcp

dm9000 i/o: 0x08000000, id: 0x90000a46
MAC: 00:14:2d:00:09:c0
operating at 100M full duplex mode
BOOTP broadcast 1
DHCP client bound to address 10.0.7.241
TFTP from server 10.0.7.1; our IP address is 10.0.7.241
```

Una vez obtenida la IP, es necesario descargar del PC de desarrollo la imagen del kernel 2.4 mediante el protocolo *tftp*, para lo cual ejecutamos el comando `tftpboot 0xa1000000 uImage24`. Esto transfiere la imagen a la dirección de memoria $0xa1000000$:

```
u-boot$ tfptpboot 0xa1000000 uImage24

dm9000 i/o: 0x08000000, id: 0x90000a46
MAC: 00:14:2d:00:09:c0
operating at 100M full duplex mode
TFTP from server 10.0.7.1; our IP address is 10.0.7.241
Filename 'uImage24'.
Load address: 0xa1000000
Loading: *T ################################################################
 ################################################################
 ################################################################
 ##############
done
Bytes transferred = 1070025 (1053c9 hex)
```

Una vez cargada la imagen en memoria, indicamos a u-boot que la arranque mediante el

comando `bootm 0xa1000000`, tras lo cual el sistema continua automáticamente con el arranque del sistema operativo y los servicios, igual que en el apartado anterior.

```
u-boot$ bootm 0xa1000000

## Booting image at a1000000 ...
   Image Name:   Linux Kernel Image
   Created:      2006-11-21  14:20:55 UTC
   Image Type:   ARM Linux Kernel Image (gzip compressed)
   Data Size:    1069961 Bytes =  1 MB
   Load Address: a0008000
   Entry Point:  a0008000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK

Starting kernel ...

Linux version 2.4.29-vrs1-pxa-intc4-col3 (root@pc7131te) (gcc version 3.3.2) #1 Tue Nov 21 14:26:35 CET 2006
CPU: XScale-Bulverde revision 7
Machine: Toradex Colibri Module
Ignoring unrecognised tag 0x00000000
Ignoring unrecognised tag 0x00000000
Ignoring unrecognised tag 0x00000000
Ignoring unrecognised tag 0x00000000
Run Mode clock: 208.00MHz (*16)
Turbo Mode clock: 312.00MHz (*1.5, active)
Memory clock: 208.00MHz (Alt=1, SDCLK[0]=/4, SDCLK[1]=/2)
System bus clock: 208.00MHz
MM: not creating mapping for 0x00000000 at 0x00000000 in user region
On node 0 totalpages: 16384
zone(0): 16384 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: root=/dev/nfs ip=:::::eth0: console=ttyS0,9600n8
Console: colour dummy device 80x30
Calibrating delay loop... 311.29 BogoMIPS
Memory: 64MB = 64MB total
Memory: 62356KB available (1954K code, 384K data, 92K init)
Dentry cache hash table entries: 8192 (order: 4, 65536 bytes)
Inode cache hash table entries: 4096 (order: 3, 32768 bytes)
Mount cache hash table entries: 512 (order: 0, 4096 bytes)
Buffer cache hash table entries: 4096 (order: 2, 16384 bytes)
Page-cache hash table entries: 16384 (order: 4, 65536 bytes)
CPU: Testing write buffer: pass
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
Init freq:312000kHz.
Registering CPU frequency change support.
CPU clock: 312.000 MHz (13.000-624.000 MHz)
CPU voltage: 1.500 mV (0.900-1.500 mV)
Register device ipmc successgul.
 Starting kswapd
Journalled Block Device driver loaded
JFFS version 1.0, (C) 1999, 2000  Axis Communications AB
JFFS2 version 2.2. (C) 2001-2003 Red Hat, Inc.
Console: switching to colour frame buffer device 80x30
pty: 256 Unix98 ptys configured
Serial driver version 5.05c (2001-07-08) with no serial options enabled
ttyS00 at 0x0000 (irq = 22) is a PXA UART
ttyS01 at 0x0000 (irq = 21) is a PXA UART
ttyS02 at 0x0000 (irq = 20) is a PXA UART
SA1100 Real Time Clock driver v1.00
<DM9000> I/O: f4000000, VID: 90000a46
tag value: 0x2d1400 0xc009
MAC: 00:14:2d:00:09:c0
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
loop: loaded (max 8 devices)
PPP generic driver version 2.4.2
Uniform Multi-Platform E-IDE driver Revision: 7.00beta4-2.4
```

```
ide: Assuming 50MHz system bus speed for PIO modes; override with idebus=xx
SCSI subsystem driver Revision: 1.00
ac97_codec: AC97 Audio codec, id: PSC4 (Philips UCB1400)
In file pxa-ac97.c, pm_registered.
UCB1x00: IRQ probe skipped, using IRQ175
Probing Colibri flash at physical address 0x00000000 (32-bit buswidth)
Colibri flash: Found 2 x16 devices at 0x0 in 32-bit bank
 Intel/Sharp Extended Query Table at 0x010A
  Unknown Intel/Sharp Extended Query version 1.4.
gen_probe: No supported Vendor Command Set found
MMC core driver installed
MMC block driver installed
Linux Kernel Card Services 3.1.22
  options:  [pm]
Intel PXA250/210 PCMCIA (CS release 3.1.22)
usb.c: registered new driver hub
host/usb-ohci.c: USB OHCI at membase 0xfe000000, IRQ 3
usb.c: new USB bus registered, assigned bus number 1
hub.c: USB hub found
hub.c: 2 ports detected
usb.c: registered new driver hiddev
usb.c: registered new driver hid
hid-core.c: v1.8.1 Andreas Gal, Vojtech Pavlik <vojtech@suse.cz>
hid-core.c: USB HID support drivers
Initializing USB Mass Storage driver...
usb.c: registered new driver usb-storage
USB Mass Storage support registered.
mice: PS/2 mouse device common for all mice
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 4096 bind 8192)
Sending DHCP requests ..., OK
IP-Config: Got DHCP answer from 10.0.7.1, my address is 10.0.7.241
IP-Config: Complete:
      device=eth0, addr=10.0.7.241, mask=255.255.255.0, gw=10.0.7.254,
     host=colibri, domain=dte.uma.es, nis-domain=(none),
     bootserver=10.0.7.1, rootserver=10.0.7.1, rootpath=/colibri
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
.....................................................................
```

5. Acceso al PC de desarrollo

El puesto de desarrollo dispone de sistema operativo Linux y contiene todas las herramientas necesarias para el desarrollo de aplicaciones para el sistema empotrado. Para acceder al PC de desarrollo se utilizará una cuenta sin privilegios, la cuenta `alumnos` con clave `dtealumnos`. Una vez abierto el entorno gráfico utilizando la cuenta sin privilegios, se puede abrir una consola de terminal en la que ejecutar comandos. Bajo ciertas condiciones, y para ejecutar determinadas operaciones, puede ser necesario ganar privilegios de superusuario en la consola de terminal, para lo que se utiliza el comando `su`. Para poder obtener estos privilegios es necesario suministrar la clave de `root`, que es `labsidi`. Se ruega no modificar estas contraseñas.

6. Acceso remoto al sistema empotrado

Además de utilizar la consola de terminal sobre el puerto serie, se puede acceder también remotamente al sistema empotrado desde el PC de desarrollo a través de la red mediante el protocolo *ssh*. El sistema empotrado tiene instalado un servidor *ssh* que permite la ejecución de una sesión de terminal remota desde otro PC. Para conectarse de esta manera desde el PC de desarrollo, basta con abrir una consola de comandos, ejecutar *ssh root@10.0.7.241* y suministrar la contraseña de superusuario del sistema empotrado. Una vez hecho esto, queda establecida

una sesión remota de ejecución de comandos en el sistema empotrado. En este punto hay que hacer notar que 10.0.7.241 es la dirección IP con la que queda configurado el sistema empotrado, mientras que el PC de desarrollo está configurado con la dirección IP 10.0.7.1.

7. Compilación y ejecución de aplicaciones de usuario

En esta sección se presenta muy brevemente el procedimiento para compilar y crear aplicaciones de usuario utilizando el compilador de C. En el PC de desarrollo se encuentran instalados dos compiladores de C: Un compilador nativo que permite crear aplicaciones ejecutables en el propio PC de desarrollo, y un compilador cruzado que permite compilar en el PC de desarrollo aplicaciones que se van a ejecutar sobre el sistema empotrado. En ambos casos el compilador utilizado es el compilador *gcc* del proyecto GNU [4], que en un caso ha sido configurado para compilar aplicaciones nativas, y en el otro para realizar la compilación cruzada.

El compilador nativo del sistema se invoca mediante el comando *gcc* cuya sintaxis no se detalla en este documento, ya que puede consultarse en la documentación de dicha aplicación y mediante el sistema de manuales de Linux. El compilador cruzado se invoca mediante el comando *arm-linux-gcc* y su sintaxis es similar a la del compilador nativo.

Veamos un ejemplo de compilación nativa y cruzada de una aplicación sencilla, el "Hola Mundo". En el PC de trabajo se ha creado un directorio con aplicaciones ejemplos (`/home/alumnos/ejemplos`) en el cuál se incluye este ejemplo. Para compilarlo y ejecutarlo en el propio PC de desarrollo, abrimos una consola de comandos, cambiamos de directorio e invocamos al compilador:

```
cd /home/alumnos/ejemplos/hola
gcc hola.c -o hola
```

Esto nos debe generar un fichero ejecutable llamado *hola*, que podremos ejecutar escribiendo:

```
./hola

Hola Mundo
```

Para compilar esta aplicación de forma cruzada, simplemente se utiliza el compilador cruzado en lugar del nativo:

```
cd /home/alumnos/ejemplos/hola
arm-linux-gcc hola.c -o hola
```

Nótese que ahora el ejecutable *hola* no puede ser ejecutado en el PC de desarrollo, ya que ha sido compilado para una arquitectura diferente. Para poder ejecutarlo, primero lo copiamos al directorio compartido utilizado por el sistema empotrado como sistema de ficheros raíz:

```
cp hola /colibri/home/root/
```

Y luego abrimos una sesión remota con el sistema empotrado de la forma que se comentó en la sección 6 (si no la tenemos ya abierta) y ejecutamos el fichero:

```
ssh root@10.0.7.241
passwd:

./hola

Hola mundo
```

Nótese que como en el PC de desarrollo se pueden abrir simultáneamente varias consolas de comando, se puede mantener una sesión remota con el sistema empotrado a la vez que se mantiene una sesión local para invocar al compilador cruzado y crear las aplicaciones.

8. Guardando el trabajo realizado

Como ya se ha mencionado anteriormente las aplicaciones se compilarán en el PC de desarrollo y se trasladarán al sistema empotrado mediante el sistema de ficheros compartido. Para guardar el trabajo realizado sobre el código desarrollado, se recomienda el uso de un pendrive USB. En esta sección se incluyen unas breves instrucciones para el uso del pendrive bajo el sistema operativo Linux instalado en los PCs del laboratorio.

Antes de utilizar un Pendrive en Linux es necesario *montarlo* en un directorio del árbol de directorios. Para ello, tras conectar el pendrive al USB se ejecutará el siguiente comando:

```
mount -t vfat /dev/uba1 /mnt/pendrive
```

De esta forma, en el directorio `/mnt/pendrive` quedará accesible el contenido del pendrive, y los archivos que se copien en dicho directorio (mediante el comando `cp`) quedarán almacenados en el pendrive. Puesto que el sistema de ficheros del pendrive es FAT32, los archivos perderán la información de permisos al ser copiados al pendrive. Para evitar esto, lo mejor es crear un fichero comprimido con el contenido del directorio de trabajo y pasar dicho fichero al pendrive, y al restaurarlo, se copia el fichero desde el pendrive y se descomprime.

El siguiente comando almacena el contenido completo de un directorio, incluyendo la información de permisos, en un fichero comprimido:

```
tar cvfz fichero.tar.gz directorio/
```

Para descomprimirlo ejecute simplemente:

```
tar xvfz fichero.tar.gz
```

Finalmente es necesario señalar que **antes** de extraer el pendrive es necesario desmontarlo emitiendo el siguiente comando:

```
umount /mnt/pendrive
```

Si se extrae el pendrive sin desmontarlo, el sistema puede volverse inestable y posiblemente el pendrive no vuelva a ser detectado correctamente hasta que se reinicie el sistema.

9. Ejemplos suministrados

En el directorio de `/home/alumnos/ejemplos` se han dejado una serie de programas que ilustran algunos aspectos de la programación avanzada en Unix, como por ejemplo la creación de procesos hijos, la comunicación entre procesos mediante señales y colas de mensajes, la comunicación mediante sockets, etc. Estos ejemplos que pueden compilarse tanto para el sistema empotrado como para el PC de desarrollo. No es objetivo de este documento realizar una introducción a la programación avanzada en UNIX, para lo cual se recomienda la consulta del excelente libro *Unix Network Programming* de Richard Stevens [5, 6]. No obstante, en la sección `documentos` del directorio `Ejemplos` se incluyen algunos tutoriales realizados por otros autores como introducción a este tema.

Dentro del directorio ejemplos se incluyen varios subdirectorios que ilustran los siguientes aspectos:

- `Procesos`. Creación de procesos hijos.
- `Fifo`. Comunicación de procesos mediante colas de mensaje.
- `Signal`. Comunicación de procesos mediante señales. Utilización de señales.
- `Socket`. Comunicación de procesos en red mediante sockets.

Todos estos ejemplos se pueden compilar de forma nativa o cruzada de la misma forma que el ejemplo de la sección 7

10. Compilación del kernel

En esta sección se describe brevemente cómo compilar e instalar un kernel para el sistema empotrado. Lo recogido en esta sección no pretende ser una guía de referencia sobre la compilación kernel de linux. Para mayor información al respecto se recomienda consultar la documentación disponible en [1], [7] y [8]

Como ya se ha mencionado anteriormente, el sistema se suministra con 2 kernels diferentes ya compilados y listos para ejecutarse. Las fuentes del kernel se encuentran almacenadas en el directorio `/usr/local/colibri-bsp-2.2/src/linux-2.4.19-vrs1-pxa1-intc4-col3` y `/usr/local/colibri-bsp-2.2/src/linux-2.6.12.4-col2`, ya que son necesarias para la compilación de drivers de dispositivos que se trata en la próxima sección. La compilación del kernel no es necesaria, pues ya se dispone de imágenes creadas, pero se incluye en esta sección por si fuese necesario volver a compilarlas para cambiar la configuración del sistema.

Para compilar el kernel, es necesario abrir una consola de comandos en el PC de desarrollo y situarse en el directorio que contiene las fuentes del kernel. Una vez hecho esto, se salva el fichero de configuración de la anterior compilación y se hace una limpieza del directorio de código fuente para eliminar ficheros temporales generados en anteriores compilaciones, y se restaura el fichero de configuración:

```
cd /usr/local/colibri-bsp-2.2/src/linux-2.6.12.4-col2
cp .config .oldconfig
make mrproper
cp .oldconfig .config
```

Una vez hecho esto se pueden reconfigurar las opciones de compilación del kernel, mediante el siguiente comando, que abre un menú de configuración a través del cual se pueden elegir diferentes opciones:

```
make menuconfig
```

Una vez configuradas las opciones de compilación y guardado el fichero de configuración, se procede a compilar la imagen del kernel, y a compilar e instalar los módulos. Las fuentes suministradas han sido parcheadas y configuradas adecuadamente de forma que directamente se compilan utilizando el compilador cruzado. Para ello se ejecutan los siguientes comandos:

```
make dep #Esto sólo es necesario en kernel 2.4
make -j2 zImage
make -j2 modules
make modules-install
```

Los módulos del kernel quedarán instalados en el directorio `/lib/modules/2.6.12.4-col2` del PC de desarrollo. Dicho directorio debe ser movido al directorio `/colibri/lib/modules/2.6.12.4-col2` para que formen parte del sistema de ficheros raíz del sistema empotrado.

La imagen del kernel queda en el fichero `vmlinux`, pero debe ser tratada un poco más de forma que pueda ser correctamente descomprimida y cargada por el gestor de arranque u-boot. Para ello se ejecuta:

```
arm-linux-objcopy -O binary -R .note -R .comment -S vmlinux linux.bin
gzip -c -9 linux.bin > linux.bin.gz
mkimage -A arm -O linux -T kernel -C gzip -a 0xa0008000 -e 0xa0008000 -n "Linux Kernel Image" -d linux.bin.gz uImage
```

Esto genera el archivo `uImage` que es una imagen del kernel cargable por el gestor de arranque u-boot. Para arrancar dicha imagen en el inicio del sistema, se almacena el fichero imagen en el directorio `/tftproot` del PC de desarrollo y se procede de forma similar al apartado 4.2.

11. Imagen y módulos del kernel

El kernel de Linux no está únicamente constituido por la imagen que se carga en el arranque, sino que existen otra serie de componentes denominados módulos que pueden cargarse o descargarse dinámicamente en tiempo de ejecución. Normalmente la imagen que se carga al comienzo contiene el núcleo del sistema operativo, es decir el código básico que permite la ejecución de diferentes tareas, así como el código necesario para la gestión del hardware más básico. Los módulos por contra, contienen código que corresponden a dispositivos que no siempre están presentes en todos los sistemas (por ejemplo, tarjetas de red, tarjetas de sonido, tarjetas de video, controladores de periféricos USB, etc.), o bien correspondiente a funciones del núcleo que no siempre se utilizan (por ejemplo soporte para determinados sistemas de fichero, etc.). Los ficheros binarios de los módulos en un sistema Linux se almacenan en un subdirectorio de `/usr/lib/modules/` que coincide en nombre con la versión del sistema kernel a la que corresponden.

La lista de módulos que el sistema tiene cargados en un momento dado puede consultarse mediante el comando `lsmod`. Para cargar un módulo en tiempo de ejecución se utiliza los comandos `modprobe` o `insmod`. Para descargar un módulo se utiliza el comando `rmmod`. Cuando se descarga un módulo se liberan algunos recursos de memoria del kernel, pero deja de poder utilizarse la funcionalidad ofrecida por el módulo. Si el módulo está siendo utilizado al ejecutar el comando `rmmod`, el módulo no se descargará.

12. Introducción a la programación de un driver de dispositivo

En esta sección se recoge brevemente el procedimiento para compilar, cargar y ejecutar un ejemplo de driver de dispositivo cuyo código se suministra como demostración. En ningún caso esta sección pretende ser una guía sobre la programación de drivers en linux. Para mayor información al respecto consulte el excelente libro *Linux Device Drivers* [9] que puede obtenerse gratuitamente en formato electrónico. El código fuente de ejemplo suministrado con dicho libro puede compilarse siguiendo el mismo procedimiento explicado en esta sección. Dicho código fuente, así como el ejemplo suministrado en esta sección corresponden a kernels de la familia 2.6, por lo que no son válidos para kernels de versión 2.4.

Un driver de dispositivo es un componente software que incluye el código necesario para la gestión del hardware, ofreciendo un interfaz de alto nivel a las aplicaciones de usuario. En Linux, los drivers se implementan normalmente como módulos del kernel, que pueden cargarse o descargarse dinámicamente en tiempo de ejecución. El interfaz entre el driver y las aplicaciones de usuario es normalmente un fichero de tipo especial (fichero de dispositivo). Una aplicación de usuario puede abrir y cerrar un fichero de dispositivo, así como leer de él o escribir como si fuese un fichero normal, siendo el driver el que implementa las funciones que se ejecutan en la apertura, cierre, lectura o escritura del fichero. Para más información consulte la referencia anteriormente mencionada [9].

En el subdirectorio `/home/alumnos/ejemplos/drivers/short-arm` se ha dejado un ejemplo de módulo del kernel para el sistema empotrado. Este módulo de ejemplo utiliza dos de los pines de propósito general del microprocesador (GPIO15 y GPIO80), a los que se tiene acceso por el conector X2 de la placa (pines 16 y 17, consúltese el datasheet). Uno de ellos (GPIO15) se configura como salida y el otro (GPIO80) como entrada capaz de producir interrupciones. El módulo utiliza como interfaz con las aplicaciones de usuario el fichero especial de dispositivos `/dev/short0`. Cuando se realiza una escritura en dicho fichero de dispositivo el módulo pondrá el pin de salida a 0 o a 1 dependiendo de si el primer byte de los datos escritos es par o impar. Cuando se realiza una lectura el proceso que lee queda bloqueado hasta que se produzca una interrupción, y una vez que esta se produce, obtendrá en la lectura el instante de tiempo en el que se produjo dicha interrupción. La salida del pin GPIO15 (pin 16 el conector X2) se puede testear mediante un osciloscopio.

Para compilar el módulo es necesario configurar adecuadamente en primer lugar una serie de variables de entorno que indican la arquitectura para la que se va a compilar, el prefijo del compilador cruzado y el directorio donde se encuentran las fuentes del kernel, y finalmente se invoca al comando *make*:

```
export ARCH=arm
export CROSS_COMPILE=arm-linux-
export KERNELDIR=/usr/local/colibri-bsp-2.2/src/linux-2.6.12.4-col2
make
```

Estos módulos del kernel sólo pueden ser cargados en el sistema empotrado, para lo cual, una vez compilado se copiará el directorio `short-arm` en el directorio compartido `/colibri`, que es el directorio raiz del sistema empotrado. Se abrirá una sesión remota con el sistema empotrado, y una vez dentro del directorio `short-arm` se ejecutará el script `short_load`, que creará los ficheros de dispositivos necesarios (`/dev/short0`) y realizará la carga del módulo. Para descargar el módulo y eliminar los ficheros de dispositivo, se utilizará el script `short_unload`.

Para probar el módulo, se pueden hacer uso de aplicaciones normales que hagan acceso a ficheros. Por ejemplo, el comando siguiente pondrá a nivel alto el pin GPIO15:

```
echo 1 > /dev/short0
```

Mientras que el siguiente comando lo pondrá a nivel bajo:

```
echo 0 > /dev/short0
```

Por otra parte el siguiente comando abre el fichero en modo lectura, quedando bloqueado a la espera de que se produzca una interrupción. Cuando esto ocurra, devolverá en pantalla el instante de tiempo en el que se ha ejecutado la rutina de atención a la interrupción:

```
cat /dev/short0
```

13. Servidor web y scripts CGI

Para finalizar se introduce brevemente en esta sección el servidor web instalado en el sistema empotrado, así como el funcionamiento de un pequeño script CGI que se ha dejado como ejemplo de aplicación de control remoto vía web. Queda fuera del ámbito de esta guía una explicación más detallada del funcionamiento de los servidores web y los fundamentos de la programación de scripts CGI.

En la plataforma empotrada se encuentra instalado el servidor web `thttp`, que se arranca al inicio del sistema. Dicho servidor permite acceder remotamente mediante un navegador a los archivos que se sitúen en el directorio `/srv/www` del sistema empotrado. Para comprobar el acceso, basta con abrir un navegador en el PC de desarrollo y especificar la URL *http://10.0.7.241*.

El servidor `thttp` permite también la ejecución de scripts CGIs en el sistema empotrado, lo cual permite la creación de aplicaciones web interactivas. Un ejemplo de apliación se ha dejado en el directorio `/srv/www/cgi-bin`, la cual puede ejecutarse remotamente especificando la siguiente URL en el navegador: *http://10.0.7.241/cgi-bin/ejemplo.cgi*. Esta aplicación permite activar o desactivar la salida GPIO15 a través del servidor web marcando o desmarcando la casilla correspondiente. Para que esta aplicación funcione es necesario que en el sistema empotrado esté cargado el módulo `short-arm` explicado en la sección anterior.

14. Aplicaciones Gráficas

El sistema empotrado dispone de una salida VGA y de un entorno gráfico para sistemas empotrados que permite el desarrollo de aplicaciones con interfaz gráfico. La interfaz gráfica sólo funciona bajo el kernel de la versión 2.4, ya que el kernel 2.6 no soporta el dispositivo gráfico. así que para poder utilizarla hay que arrancar el sistema empotrado con el kernel 2.4 (véase sección 4.2).

El entorno gráfico utilizado es el *Opie* [10], que es un entorno gráfico ligero para PDAs basado en las librerías QTE de Trolltech [11]. Estas librerías son una versión reducida, orientada a sistemas empotrados, de las librerías QT con las que están desarrollada el conocido entorno gráfico de Linux KDE. Las librerías QTE además acceden directamente al *framebuffer* del dispositivo, eliminando la necesidad de librerías gráficas adicionales como las librerías X. La aplicaciones gráficas basadas en la librería QTE se desarrollan en C++ haciendo uso de las clases de la librería, y son perfectamente soportadas por el compilador cruzado de c++ que está instalado en

el PC de desarrollo (`arm-linux-g++`). La información necesaria para el desarrollo y compilación de estas aplicaciones y tutoriales de ejemplo pueden encontrarse en [10] y [12].

Para poder utilizar el entorno gráfico es necesario un dispositivo *señalador*, lo cual puede conseguirse fácilmente conectando un ratón USB al sistema empotrado.

15. Bluetooth

En el sistema empotrado Bluetooth es soportado, al igual que en todos los sistemas Linux, mediante un conjunto de módulos del kernel, librerías y herramientas conocido como Bluez [13]. Para poder utilizar estas herramientas bastará con conectar un *dongle* Bluetooth USB al sistema empotrado. En este documento sólo se incluyen algunos ejemplos sencillos de utilización de la herramienta, pues no es objetivo de este documento dar detalles de cómo realizar la configuración de bluetooth bajo Linux, ni cómo se realizaría la programación de aplicaciones utilizando esta tecnología. Para ello se recomienda consultar la información disponible en [14].

Para arrancar e inicializar el módulo bluetooth, se ejecutaría:

```
hciconfig hci0 up
```

Una vez arrancado los servicios bluetooth, para hacer un *inquiry* (detección de dispositivos vecinos), se ejecutaría el siguiente comando:

```
hcitool inq
```

Referencias

[1] "The linux documentation project." [Online]. Available: http://www.tldp.org

[2] K. Yaghmour, *Building Embedded Linux Systems*. O'Reilly, 2003.

[3] "Openembedded project." [Online]. Available: http://www.openembedded.org

[4] "The gnu c compiler project." [Online]. Available: http://www.gcc.org

[5] R. Stevens, *UNIX Network Programming. Vol. 1: Networking API*. Prentice Hall, April 1998, vol. 1.

[6] ——, *UNIX Network Programming. Vol. 2: Interprocess Communication*. Prentice Hall, April 1998, vol. 2.

[7] "Linux hearquarters." [Online]. Available: http://www.linuxHQ.com

[8] "Kernel newbies." [Online]. Available: http://www.kernelnewbies.org/

[9] J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers. Third Edition*. O'Reilly, 2004.

[10] "Opie desktop project." [Online]. Available: http://opie.handhelds.org/

[11] "Trolltech." [Online]. Available: http://www.trolltech.com/

[12] “Qt-embebido.” [Online]. Available: http://doc.trolltech.com/2.3/

[13] “Bluez: Official linux bluetooth protocol stack project page.” [Online]. Available: http://www.bluez.org

[14] M. Holtmann, “Blueooth and linux.” [Online]. Available: http://www.holtmann.org/linux/bluetooth/