



FACULTAD
DE INGENIERÍA
Universidad de Buenos Aires

Implementación de manejadores de dispositivos

— Presentación del Trabajo Final —

Alumno: Esp. Ing. Rodrigo Tirapegui

Beagle Bone Black



- ▶ SoC Texas Instruments AM335x (ARM Cortex-A8 CPU)
- ▶ 512 MB of RAM
- ▶ 4 GB of on-board eMMC storage
- ▶ USB host and USB device, microSD, micro HDMI
- ▶ Conexión Ethernet
- ▶ 2 x 46 pins headers, with access to many expansion buses (I2C, SPI, UART)



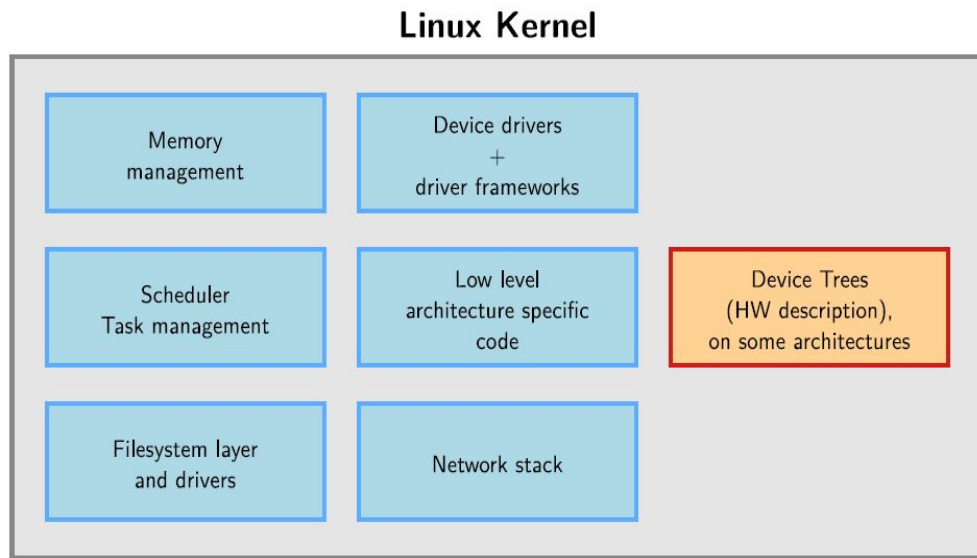
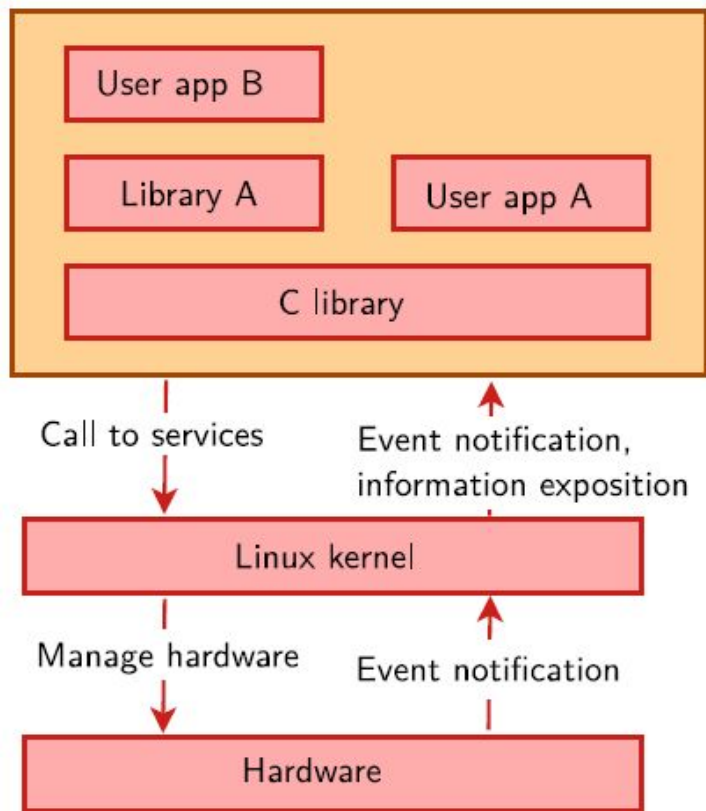
Características del kernel linux

- ▶ Es altamente portable a múltiples hardwares y corre en la mayoría de las arquitecturas (de 32-bits mínimo, con/sin MMU).
- ▶ Es fácil de programar y abunda la información en Internet.
- ▶ Es seguro porque su código es revisado por una comunidad de expertos. Security. It can't hide its flaws. Its code is reviewed by many experts.
- ▶ Es estable y confiable.
- ▶ Es modular: se puede compilar tan solo con lo que el sistema en que se usará necesita.


Características del kernel linux

- ▶ Maneja todos los recursos de hardware: CPU, memory, I/O.
- ▶ Provee un conjunto de interfaces (System calls) que le permiten a aplicaciones de usuario y librerías acceder a los recursos de hardware.
- ▶ Se encarga de controlar el acceso concurrente al hardware desde múltiples aplicaciones.

Características del kernel linux



Compilando el kernel linux

 *Obtención de la versión mainline del kernel 4.19.y*

```
$ git clone http://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
```

```
$ cd ~/linux-kernel-labs/src/linux/
```

```
$ git remote add stable
```

```
git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
```

```
$ git fetch stable
```

```
$ git checkout -b 4.19.y stable/linux-4.19.y
```

Compilando el kernel

▶ *Instalación del compilador*

```
$ sudo apt install gcc-arm-linux-gnueabi
```

▶ Se configuran los fuentes del kernel con la configuración por defecto para las placas OMAP2 y la familia AM335x a la que pertenece la BeagleBone.

▶ Se agregan las variables de entorno ARCH=arm y CROSS_COMPILE=arm-linux-gnueabi-

▶ Se corren múltiples tareas en paralelo para acelerar la compilación

```
$ make -j 8
```

Arrancando el kernel

▶ *Instalación del paquete servidor NFS y reinicio*

```
$ sudo /etc/init.d/nfs-kernel-server restart
```

▶ *Reinicio de la BeagleBone y login siguiendo los pasos:*

1. Cargar zImage a partir de la dirección de memoria 0x81000000
2. Cargar am335x-customBoneBlack.dtb a partir de la dirección de memoria 0x82000000
3. Arrancar elkernel con bootz 0x81000000 - 0x82000000

Arrancando el kernel

 *En la terminal de UBoot ejecutamos la secuencia*

```
$ picocom -b 115200 /dev/ttyUSB0
```

```
$ env default -f -a
```

```
$ setenv ipaddr 192.168.0.100
```

```
$ setenv serverip 192.168.0.1
```

```
$ setenv bootargs root=/dev/nfs rw ip=192.168.0.100 console=ttyO0,115200n8  
nfsroot=192.168.0.1:/home/rodrigo/linux-kernel-labs/modules/nfsroot,nfsvers=3
```

```
$ tftp 0x81000000 zImage
```

```
$ tftp 0x82000000 am335x-customboneblack.dtb
```

```
$ bootz 0x81000000 - 0x82000000
```

Generando un device tree a medida

Device tree I2C1

```
&i2c1 {  
    status = "okay";  
    pinctrl-names = "default";  
    clock-frequency = <100000>;  
    pinctrl-0 = <&i2c1_pins>;  
  
    myMPU9250: myMPU9250@68 {  
        compatible = "mse,myMPU9250";  
        reg = <0x68>;  
    };  
};
```

Editamos el Makefile agregando

```
am335x-customboneblack.dtb \
```

Compilamos el device tree

```
$ make dtbs
```

Copiamos el .dtb generado al server home directory del server TFTP

Escribiendo un módulo de kernel

▶ Se “linkea” el device tree con el driver asociado al dispositivo

```
static const struct of_device_id myMPU9250_of_match[] =  
{  
    { .compatible = "mse,myMPU9250" },  
    { }  
};
```

```
MODULE_DEVICE_TABLE(of, myMPU9250_of_match);
```

Escribiendo un módulo de kernel

▶ Se definen los dispositivos soportados por el driver

```
static const struct i2c_device_id myMPU9250_i2c_id[] =  
{  
    { "myMPU9250", 0 },  
    { }  
};
```

```
MODULE_DEVICE_TABLE(i2c, myMPU9250_i2c_id);
```

Escribiendo un módulo de kernel

▶ Se asignan los punteros a función que serán invocados por el core I2C para cada dispositivo soportado por el driver

```
static struct i2c_driver myMPU9250_i2c_driver =
{
    .driver =
    {
        .name = "myMPU9250",
        .of_match_table = myMPU9250_of_match,
    },
    .probe = myMPU9250_probe,           /* Inicializa el dispositivo      */
    .remove = myMPU9250_remove,        /* Remueve el dispositivo        */
    .id_table = myMPU9250_i2c_id       /* Tabla de dispositivos soportados */
};

module_i2c_driver(myMPU9250_i2c_driver);
```

Character device - concepto

- ▶ Permite intercambiar información hacia y desde una aplicación que corre en el espacio de usuario mediante **Open, Read, Write, Close**, etc.
- ▶ Son en esencia un framework para que aplicaciones de usuario accedan a hardware a través de diferentes drivers (serial, video, audio, etc) de dispositivos listados en **/dev**.
- ▶ Se identifican con dos números *Major* y *Minor*.
- ▶ Cada driver registra callback para implementar las operaciones mencionadas mediante una estructura de tipo **file_operations**.

Character device - intercambio de datos

▶ El driver implementa los callback de las siguientes *file_operations*:

```
static struct file_operations fops =  
{  
    .open = dev_open,  
    .read = dev_read,  
    .write = dev_write,  
    .release = dev_release,  
};
```

Character device - intercambio de datos

▶ *El intercambio de datos entre user y kernel space se hace mediante las funciones:*

```
copy_to_user(*to, *from, n);
```

```
copy_from_user(*to, *from, n);
```


Character device - creación y destrucción

▶ *Utiliza las siguientes funciones de creación y destrucción:*

```
register_chrdev(major, name, fops); /* Registra el character device */  
  
class_create(owner, name);      /* Registra el objeto del dispositivo */  
  
device_create(class, parent, devt, fmt,...); /* Crea archivo en /dev */  
  
device_destroy(class, devt); /* Libera el archivo de /dev */  
  
class_unregister(class); /* De-registra el objeto del dispositivo */  
  
class_destroy(class); /* Libera el espacio alocado por el dispositivo */  
  
unregister_chrdev(major, name); /* De-registra el character device */
```

Compilando el linux kernel module (LKM)

▶ Se compila el LKM con el comando \$ make y se instala en la BeagleBone ejecutando desde el UBoot prompt la sentencia:

```
# insmod <module_name>.ko
```

▶ Se remueve el módulo ejecutando desde el UBoot prompt la sentencia:

```
# rmmod <module_name>
```

¡Gracias por su atención!