

First European Laboratory on  
**Real-Time and Control**  
Pisa, 10th – 14th July 2006



## S.Ha.R.K. – Drivers & Interrupts

**Mauro Marinoni** (mauro.marinoni@unipv.it)

### Robotic Lab

University of Pavia (Italy)



**ARTIST2**



## Objectives

- Drivers realization
  - Different approaches
  - S.Ha.R.K. evolution
- Present implementation in S.Ha.R.K.
  - Interrupt Server
  - Linux Compatibility Layer
  - Available drivers
- Work in progress
- Future Work

Pisa, 11th July 2006

2 / 55



## Drivers Implementation

- Different approaches to a new driver layer implementation:
  - To write it from scratch ;
  - To reuse low level calls code and rewrite the rest;
  - To reuse as much code as possible;
- Most of the recyclable code came from NOT Real-Time Operating Systems.

Pisa, 11th July 2006

3 / 55



## Comparison Terms

- The approach must be selected according to:
  - Requirements:
    - Predictability;
    - Reliability;
    - Stability;
    - Easy Maintenance.
  - Team characteristics:
    - Dimension;
    - Knowledge;
    - Time.

Pisa, 11th July 2006

4 / 55



## Advantages & Drawbacks

- Importing pre-existent code:
  - It increase Reliability and Stability;
  - It require less technical knowledge and production time;
  - It decrease the amount of code that must be implemented.

Pisa, 11th July 2006

5 / 55




## Advantages & Drawbacks

- The code is not developed focusing on real-time issues, then predictability is penalized.
- The imported code usually presents a lot of features unusable on the new OS. This increases the execution time and the total code dimension.

*"The only piece that can't break off is the one you don't mount"*  
Spitfire engineer

Pisa, 11th July 2006

6 / 55




## S.Ha.R.K. Drivers Evolution

1.22 → 1.4

- Most of the S.Ha.R.K. drivers system has been rewritten in the transition between versions 1.22 and 1.4.
- The point of view has changed from an almost homemade implementation to an integration of drivers inherited from an external O.S.

Pisa, 11th July 2006 7 / 55




## S.Ha.R.K. Drivers Evolution

1.22 → 1.4

- Drivers written from scratch;
- Almost full Real-Time implementation;
- Small drivers either in term of dimension or execution time;
- Few supported peripherals.

- Drivers inherited from Linux;
- Drivers rich in features;
- Small amount of code needs to be rewritten;
- Higher number of supported peripherals.

Pisa, 11th July 2006 8 / 55

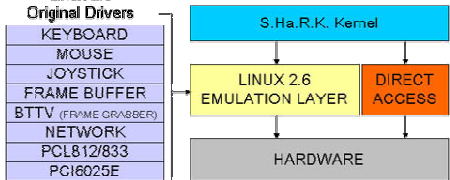


## Modular Structure


- A modular approach has been implemented in order to maintain backward compatibility and to allow full homemade drivers.

Linux 2.6  
Original Drivers

KEYBOARD  
MOUSE  
JOYSTICK  
FRAME BUFFER  
BTTV (FRAME GRABBER)  
NETWORK  
PCL812/833  
PCI6025E

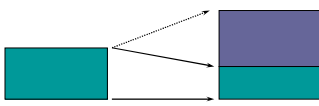


Pisa, 11th July 2006 9 / 55




## Good Things...

- The most interesting part of the new approach is that porting a new driver is quicker and it requires a smaller amount of time and code.
- I.E. Porting the whole input layer (keyboard, mouse, joystick and PC speaker)
  - It requires one person for less than a week;
  - The code dimension grows from 180Kb to 450Kb but only 70Kb need to be written.



Pisa, 11th July 2006 10 / 55




## ... and Dark Shadows

- Linux ISRs are developed to work with disabled interrupts.
- Even if we can estimate a WCET for every ISR, it is not possible to guarantee how much utilization is stolen by ISR execution time during interrupt burst.

*"One layer to rule them all, one layer to find them all, one layer to bring them all and in an utilization estimation bind them all."*


Pisa, 11th July 2006 11 / 55



## The Interrupt Server

- It is composed basically of 4 elements:
  - A scheduling algorithm thought to manage only one non-preemptable task;
  - A not-preemptable task in charge of executing handlers.
  - A FIFO queue of interrupts waiting to be processed;
  - A list of couples (interrupt number, handler function);


Pisa, 11th July 2006 12 / 55



## The Interrupt Server

- When an interrupt is raised by a peripheral:
  - The S.Ha.R.K. handler puts it in the queue and makes a request for an activation of the task;
  - If the task presents enough capacity the ISR is executed, otherwise the handler waits for the next recharge.
  - When the task ends an ISR, if some capacity is left, it looks for a new job in the queue.

Pisa, 11th July 2006 13 / 55




## The Interrupt Server

- With this approach we are able to:
  - Protect the ISR from Linux code;
  - Guarantee a maximum value for the utilization code.

*"A beginning is a very delicate time"*  
Frank Herbert, Dune


Pisa, 11th July 2006 14 / 55



## Drivers from Linux 2.6

- Input
  - Standard keyboard (Linux 2.6 Input/Keyboard driver)
  - Standard PS/2 mouse (Linux 2.6 Input/Mouse driver)
  - Analog and digital joystick (Linux 2.6 Input/Gameport)
- Graphics
  - VESA 2.0
  - MATROX, NVIDIA, ATI (only the Linux 2.6 supported) (Linux 2.6 FRAMEBUFFER driver )


Pisa, 11th July 2006 15 / 55



## Drivers from Linux 2.6

- Frame Grabbers
  - BookTree chipset BT848/BT878 (Linux 2.6 BTTV driver )
- Voltage/Frequency scaling (Linux 2.6 CPUFreq)
- USB **[Experimental]**
  - USB2.0 (ehci) and USB1.1 (uhci & ohci);
  - Keyboard, Mouse, Joystick (Human Interface Device)
  - Webcam (PWC chip)
  - Serial Adapter (PL2303 chip)


Pisa, 11th July 2006 16 / 55



## Drivers from S.Ha.R.K 1.22

- I/O PORTS
  - Standard RS232 and Parallel port
- I/O Cards
  - PCI6025E, PCL812, PCL833 (Shark 1.22 drivers patched for Linux 2.6 emulation layer)
- Network (only UDP support)
  - RTL8139, 3C90X, 3C509, PRO100 (Old Linux drivers patched to work with Linux 2.6 emulation layer )


Pisa, 11th July 2006 17 / 55



## Linux Compatibility Layer:

- Works as a filter between the S.Ha.R.K. kernel and Linux drivers;
- Remaps Linux system call to S.Ha.R.K. ones;
- Initializes Linux infrastructures like busses, classes and devices;
- Is in charge of maintaining data structures needed by drivers.


Pisa, 11th July 2006 18 / 55



## LinuxComp - Example

- A Linux driver calls the `request_irq` function:
  - Parameters are stored in a structure;
  - The `handler_set` function is executed with 2 parameters:
    - Interrupt number;
    - `linux_intr` function, from LinuxComp, as handler;
  - Return values and error codes are translated and sent back to the caller;


Pisa, 11th July 2006 19 / 55



## LinuxComp – Ex(2) (The Revenge)

- When an interrupt arises the S.Ha.R.K. kernel executes the `linux_intr` function with the interrupt number as the only parameter;
- The `linux_intr` function gets data previously sent by `request_irq` and it runs the correct handler with related parameters;


Pisa, 11th July 2006 20 / 55



## Drivers Dependencies

- S.Ha.R.K. 1.4 DOESN'T already manage all dependencies;
- For a correct application execution an order must be followed in driver initialization;
- Even when the system will exploit automatically drivers dependencies, a full list of init functions calls leads to a better understanding of the application.

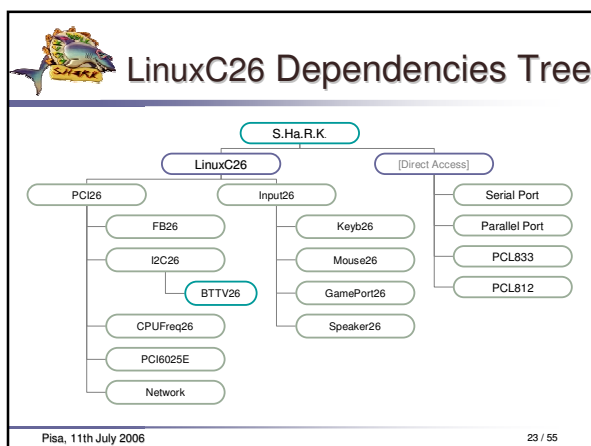

Pisa, 11th July 2006 21 / 55



## Drivers Dependencies - Ex

- The network layer needs the PCI bus drivers to probe and to initialize correctly NIC boards;
- If the PCI driver is not present when the network driver tries to scan for cards the user receives a "No network card found" message.

Pisa, 11th July 2006 22 / 55

## Drivers Functions

- Each driver presents:
  - Only one initialization function;
  - One closing function (not all of them);
  - Some control functions;
  - Some I/O functions.

Pisa, 11th July 2006 24 / 55



## LinuxC26

- LinuxC26 is the core of the compatibility layer. It presents only the initialization function: `LinuxC26_init()`.
- The only parameter decides whether the compatibility layer must use the `IntDrive` to serve interrupt requests or manage them with a fast handler.

Pisa, 11th July 2006

25 / 55



## PCI26

- The PCI26 driver needs to be initialized with the `pci26_init()` function which fills all structures for the PCI bus;
- This is important because these structures are needed by some Linux drivers like NIC, frame grabber, etc.

Pisa, 11th July 2006

26 / 55



## Input26:

- Initializes the Input layer which works as a bridge between low level drivers (i8042, atkbd, ns558, ...) and high level handlers (keyboard, mouse, joystick, ...);
- Presents only 2 functions:
  - `INPUT26_init()` – Which fills all structures and runs some chipset probes;
  - `INPUT26_close()` – That frees structures, IRQ, etc.

Pisa, 11th July 2006

27 / 55



## Keyboard26

- The driver activation is managed by 5 functions:
  - `KEYB26_init` – Which performs Hw/Sw initialization;
  - `KEYB26_close` – Which stops the software and frees hardware resources;
  - `KEYB26_installed` – That returns if the driver is installed;
  - `KEYB26_enable`, `KEYB26_disable` – Are in charge of swap between accept and drop the policy about data coming from the keyboard;

Pisa, 11th July 2006

28 / 55



## Keyboard26 - Acquisition

- The application can recover an ASCII code related to a pressed key with the `keyb_getch` function;
- If the buffer is empty, the function waits for a new key or returns 0 depending on the parameter `wait` given by the application.

Pisa, 11th July 2006

29 / 55




## Keyboard26 - KeyEvent

- It is a structure containing the following fields for each event related with keys:
  - The ASCII code;
  - The scan code;
  - The status: the key is pressed, repeated or released;
  - The flags encoding: it says if ALT, SHIFT or CTRL buttons are pressed;

Pisa, 11th July 2006

30 / 55



## Keyboard26 - KeyEvent

- As well as element by element the structure could be filled using the `setevt` macro, from version 1.5.1.

```

ev.ascii = 'k';
ev.scan = KEY_K;
ev.status = KEY_PRESSED;
ev.flag = CNTR_BIT;

```

Standard extended mode


New compact mode

```

set_keyevt (&ev, 'k', KEY_K, CNTR_BIT, KEY_PRESSED);

```


Pisa, 11th July 2006 31 / 55



## Keyboard26 - KeyEvent

- It can be used in 2 ways:
  - Directly by the `keyb_getcode` function which returns the first unprocessed event in the FIFO queue;
  - Linking an handler to a predefined KeyEvent using the `keyb_hook` function: when a new event arrives it is compared to the hooks list and, if a match is found, the related handler is executed.

Pisa, 11th July 2006 32 / 55




## Mouse26

- The mouse driver presents a structure similar to the keyboard one:
  - Initialization: `MOUSE26_init`, `MOUSE26_close`, `MOUSE26_installed`, etc.
  - Interaction with applications: `mouse_getposition`, `mouse_hook`, etc.
- There is also a set of functions for the cursor visualization either in text or graphics:
 

`mouse_on`, `mouse_off`, `mouse_txtcursor`,  
`mouse_grxcursor`, `mouse_textshape`,  
`mouse_grxshape`.


Pisa, 11th July 2006 33 / 55



## Gameport26

- The gameport driver works **ONLY** with hardware which presents a "*SoundBlaster Gameport Emulation*" on 201H port.
- The interface is composed only by:
  - Init functions: `joy26_init`, etc;
  - Activity functions: `joy_enable`, `joy_disable`;
  - Pointer position functions: `joy_setstatus`, `joy_getstatus`.


Pisa, 11th July 2006 34 / 55



## Speaker26

- Last but not least, the speaker driver allows to emit a sound through the PC speaker.
- Besides than usual function for initialization, we can see 2 functions:
  - `speaker_sound` – Which emits a beep at the required function for a chosen period or in a never ending loop;
  - `speaker_mute` – That resets the speaker output.
- It is sometimes useful for debug purposes instead of text, graphic or network debug.

Pisa, 11th July 2006 35 / 55



## FrameBuffer26

- The driver allows us to work with graphic primitives on *16 bpp SVGA* graphics modes.
- The correct order for initialization is:
  - Init hardware and internal structures with `FB26_init`;
  - Open the frame buffer using the `FB26_open` function;
  - Connect the drawing library through the `FB26_use_grx` call;
  - Set the required graphic mode with `FB26_setmode`.

Pisa, 11th July 2006 36 / 55



## FrameBuffer26

- Drawing functions are inherited from the old (S.Ha.R.K. 1.22) Grx library, maintaining the same syntax;
- Modes are in the form "widthxheight-bpp" (Ex. "640x480-16"). It can sometime be useful to fix the refresh rate manually (Ex. "640x480-16@60").

Pisa, 11th July 2006

37 / 55



## FrameGrabber

- It is divided into 2 parts:
  - The low level driver (BTTV) which manages acquisition cards based on BookTree chips;
  - The high level driver (VIDEODEV) that controls video peripherals.
- Adding another frame grabber consists into porting ONLY the new low level driver.

Pisa, 11th July 2006

38 / 55



## FrameGrabber – BTTV

- It works as a filter between the videodev interface and acquisition boards based on BookTree BT8x8 chips.
- It supplies only 2 functions:
  - BTTV26\_init – Which initializes the acquisition board and all internal structures;
  - BTTV26\_close – That shutdowns the low level driver.

Pisa, 11th July 2006

39 / 55



## FrameGrabber – VideoDev

- This layer allows an easy and uniform interaction between the application and low level drivers;
  - The function VIDEODEV26\_open opens the communication with the device through the low level driver;
  - Every other interaction is performed by using the VIDEODEV26\_ioctl1 function which sends commands to the device.

Pisa, 11th July 2006

40 / 55



## FrameGrabber – VideoDev

- The list of supported commands is the same of Linux 2.6 VideoDev with the exception of VIDIOSYNC which has been totally rewritten. It accepts a task PID as an argument; this is executed when a new frame is ready.
- Attention: The task MUST be an aperiodic one.

Pisa, 11th July 2006

41 / 55



## FrameGrabber – Example<sub>1</sub>

```
/* Init videodev driver */
VIDEODEV26_open(FRAME_GRABBER_NUMBER);


/* Select the input channel */
res = VIDEODEV26_ioctl1(FRAME_GRABBER_NUMBER,VIDIOCCHAN,(unsigned long)&chan);
chan.channel = channel;
chan.type = VIDEO_VC_TUNER;
chan.norm = VIDEO_TYPE_CAMERA;
res = VIDEODEV26_ioctl1(FRAME_GRABBER_NUMBER,VIDIOCCHAN,(unsigned long)&chan);

/* Select palette and depth */
res = VIDEODEV26_ioctl1(FRAME_GRABBER_NUMBER,VIDIOCPICT,(unsigned long)&vpic);
vpic.palette = VIDEO_PALETTE_RGB24;
vpic.depth = 24;
vpic.brightness = 35000;
vpic.hue = 32000;
vpic.contrast = 32000;
vpic.colour = 32000;
res = VIDEODEV26_ioctl1(FRAME_GRABBER_NUMBER,VIDIOCPICT,(unsigned long)&vpic);
```

Pisa, 11th July 2006

42 / 55





## FrameGrabber – Example<sub>2</sub>

```

/* Enable the tuner */
tuner.tuner = 0;
tuner.mode = VIDEO_MODE_PAL;
res = VIDEODEV26_ioctl(FRAME_GRABBER_NUMBER,VIDIOCSTUNER, (unsigned long)&tuner);


/* Set window dimensions */
res = VIDEODEV26_ioctl(FRAME_GRABBER_NUMBER,VIDIOCGWIN, (unsigned long)&win);
win.x = 0;
win.y = 0;
win.width = FG_W;
win.height = FG_H;
res = VIDEODEV26_ioctl(FRAME_GRABBER_NUMBER,VIDIOCSWIN, (unsigned long)&win);

/* Set the buffer */
res = VIDEODEV26_ioctl(FRAME_GRABBER_NUMBER,VIDIOCSFBUF, (unsigned long)(fbuf));

/* IMPORTANT: Set the aperiodic elaboration task */
VIDEODEV26_ioctl(FRAME_GRABBER_NUMBER,VIDIOCSYNC, (unsigned long)(task_PID));

```


Pisa, 11th July 2006 43 / 55



## CPUFreq26

- The driver tests the system on processor information and supplies an API interface for frequency scaling, if supported from CPU.
- A set of functions for getting scaling information is implemented: min/max frequency, current frequency, all supported frequencies, latency.
- A function permits to change the current frequency.


Pisa, 11th July 2006 44 / 55



## USB26 [Experimental]

- A single function start core, hub and HID drivers: `USB26_init()`;
- The USB layer can be closed with the `USB26_close()` function;
- The HotPlug features is **NOT** supported yet;
- Specific set of functions are used for webcams and serial adapters.


Pisa, 11th July 2006 45 / 55



## Network

- The driver is inherited from previous versions of S.Ha.R.K. and mixes homemade code with Linux 2.0 code;
- It support only a small set of NIC cards;
- It supplies IP and UDP protocols interfaces;

Pisa, 11th July 2006 46 / 55



## Network – I.E. Init

- To initialize the network subsystem the procedure is:


```

struct net_model m = net_base;

/* Set a task for TX mutual exclusion */
net_setmode(m, TXTASK);
/* Use UDP/IP stack */
net_setudpip(m, localipaddr, "255.255.255.255");
/* Start NetLib */
if (net_init(&m)!=1)
    exit(1);

```

Pisa, 11th July 2006 47 / 55



## Network – I.E. Receive

- To receive data via UDP:

```

UDP_ADDR from, local;
int s, n;

/* Connect on local port 100 */
Local.s_port = 100;
s = udp_bind(&local, NULL);

/* Receive the packet (block until it arrives) */
n = udp_recvfrom(s, n, &from);

```

Pisa, 11th July 2006 48 / 55





## Network – I.E. Send (1)

- To send data using UDP:

```
UDP_ADDR to, local;
IP_ADDR bindlist[5];
int s;

/* Set the source port */
ip_str2addr(localipaddr, &(local.s_addr));
local.s_port = 101;
```

Pisa, 11th July 2006

49 / 55



## Network – I.E. Send (2)

```
/* Bind */
ip_str2addr(remoteipaddr, &(bindlist[0]));
memset(&(bindlist[1]), 0, sizeof(IP_ADDR));
s = udp_bind(&local, NULL);

ip_str2addr(remoteipaddr, &(to.s_addr));
to.s_port = 100;

/* Send Packet */
udp_sendto(s, buff, strlen(buff), &to);
```

Pisa, 11th July 2006

50 / 55



## Putting everything on the road...

- Usually all init and close functions reside in the initialization file.
- The standard implementation is composed by:
  - One boot function which makes all requested drivers starting;
  - One function starting when the system enters the `SHUTDOWN` runlevel;
  - One aperiodic task, started by the previous function, that executes all closing actions.

Pisa, 11th July 2006

51 / 55



## How to port a driver?

- There is not a standard procedure to port a Linux driver in S.Ha.R.K. because everything depends on:
  - Underlying hardware;
  - Previously ported drivers;
  - Requested resources.
- For example:
  - USB keyboard and mouse use the input layer and don't need a user API;
  - Serial mouse searches the `tty` interface and it is not actually ported because the `tty` layer is big and is not needed by any other driver;

Pisa, 11th July 2006

52 / 55



## How to port a driver?

- Some useful tips commonly applied till now:
  - To copy Linux files in a new directory under drivers;
  - To create a new makefile (see `input/bttv/cpu/fb`);
  - To create an include directory for files needed by S.Ha.R.K. applications;
  - Sometimes to create a new directory for S.Ha.R.K. code is useful.

Pisa, 11th July 2006

53 / 55



## How to port a driver?

- Pay attention to the fact that:
  - No device filesystem is implemented: you need to create some API functions in order to obtain communication between driver and application;
  - No up/down function remapping is done by now: you need to comment them out.

Pisa, 11th July 2006

54 / 55



## I want MY driver!

- If, for some particular real-time constraints, a full real-time driver is required, then:
  - Write it from scratch;
  - Put it side by side with Linux-ported drivers;
  - Let then run together;

"In a world of fugitives, the person taking the opposite direction will appear to run away"

T.S. Elliot

Pisa, 11th July 2006

55 / 55



## TODO ?!?

- Complete drivers dependencies check: autoload a driver if requested during the boot of another driver;
- In various different developing states are:
  - USB devices:
    - Pwc webcams;
    - Gps modules;
    - Serial adapter;
    - Memory Stick (?!?)
  - Network (from Etherboot):
    - Low level drivers

Pisa, 11th July 2006

56 / 55

Mauro Marinoni [mauro.marinoni@unipv.it]

Robotic Lab - University of Pavia (Italy)



## S.Ha.R.K. – Drivers & Interrupts

- Site: <http://shark.sssup.it>
- Forum: <http://feanor.sssup.it/retis-projects>
- Bugzilla: <http://lancelot.sssup.it/bugzilla>
- Retis: <http://retis.sssup.it>
- RoboLab: <http://robot.unipv.it>