

# Feedback Scheduling in S.Ha.R.K: a First Approach

Josep Guàrdia, Pau Martí, Manel Velasco and Rosa Castañé

Automatic Control Department, Technical University of Catalonia

Research Report: ESII-RR-06-13

6th September 2006

## **Abstract**

S.Ha.R.K. is a real-time kernel developed at the Scuola Superiore S. Anna, in Pisa, Italy. It is highly modular and allows easy configuration of the scheduling policy. However, if feedback scheduling approaches aimed at selecting specific tasks periods according to application performance metrics have to be used, S.Ha.R.K. does not provide an easy interface.

In this report we describe small modifications made in the kernel. These modifications allow testing feedback scheduling policies by permitting the dynamic insertion of any optimization function together with a scheduling policy. In this way, periods will be set according to the optimization procedure, and then tasks will be scheduled according to any traditional scheduling policy.

**Disclaimer** - SUPPORT WILL NOT BE PROVIDED.

The information in this report is for informational purposes only. And the provided software is untested and non reliable. It has been used for internal purposes only.

## Contents

<b>1</b>	<b>Summary</b>	<b>4</b>
<b>2</b>	<b>Kernel modifications</b>	<b>5</b>
<b>3</b>	<b>Installation</b>	<b>6</b>
3.1	Debian installation . . . . .	6
3.2	GCC 3.3.5 Installation . . . . .	6
3.3	S.Ha.R.K. Installation . . . . .	7
3.4	Bochs Installation . . . . .	7
3.5	Booting S.Ha.R.K. HOWTO . . . . .	8
3.6	S.Ha.R.K for Dummies . . . . .	9
3.7	S.Ha.R.K for even preDummies . . . . .	9
<b>4</b>	<b>Conclusions</b>	<b>12</b>

## 1 Summary

S.Ha.R.K. [1] is a real-time kernel developed at the Scuola Superiore S. Anna, in Pisa, Italy. It is highly modular and allows easy configuration of standard scheduling policies. However, if feedback scheduling approaches aimed at selecting tasks execution patterns according to application performance metrics have to be used, S.Ha.R.K. does not provide an easy interface. Feedback scheduling for control applications refers to those flexible and adaptive scheduling or resource allocation techniques that specify controllers executions as a function of the behavior of the controlled plants. See [2], [3] or [4] for several examples of feedback scheduling for control applications.

Focusing on feedback scheduling for control tasks, there are different methods for specifying control tasks execution patterns. Here we focus on selecting tasks periods. An easy approach is to apply an optimization procedure (for example to a given a cost function on the plants states and subject to the schedulability constraints) that will set the optimal periods for the executing tasks. In order to call an optimization function from the kernel, it is compulsory to slightly modify the kernel. This modification will permit to easily and dynamically insert functions into the kernel.

This report documents these modifications, as well as, gives a short guide for quick installation and utilization. The installation assumes that Debian [5] linux is the base operating system, that includes Bochs x86 PC Emulator [6]. Bochs is a portable x86 PC emulation software package that emulates enough of the x86 CPU, related AT hardware, and BIOS to run Windows, Linux, \*BSD, Minix, and other OS's, all on your workstation. Bochs virtual machine will allow working on S.Ha.R.K. from Debian without having to reboot the systems for every run.

Finally, for quick S.Ha.R.K. utilization, a Knoppix [7] ISO bootable for a CD, with all the necessary stuff (kernel and PC emulator), is also described.

## 2 Kernel modifications

The kernel main modification permits declaring a function that will be executed every time the scheduler is about to be executed. Here, scheduler refers to the main scheduler, not to the specific scheduling policies that can be added in a modular fashion (EDF, RR, ...) <sup>1</sup>.

All modifications are listed next:

- **./shark/include/kernel/func.h**

A function pointer has been added at the end of the file

```
1 void (*preScheduler)()
```

- **./shark/kernel/kern.c**

In line 270 of function **void scheduler(void)**, the following line has been added.

```
if(*preScheduler) preScheduler()
```

- **./shark/demos/cs2/cstest.c**

We define the feedback scheduler function, namely **funcio()**

```
void funcio() {  
    cprintf("preScheduler() - %s\n", now());  
}
```

This function is inserted into the kernel using the defined pointer

```
preScheduler = funcio;
```

Note that the definition of **funcio()** will specify a particular feedback scheduling policy. Here only the mechanisms for inserting this function has been reported. It is left to the user to define the specific function.

---

<sup>1</sup>All the modifications can be found in <http://www.upcnet.es/~pmc16/all.tar>

## 3 Installation

Next, the installation, configuration and booting of S.Ha.R.K. are described in 5 steps<sup>2</sup>. It is assumed that the companion operating systems is Debian linux.

### 3.1 Debian installation

1. Debian installation can be done in different ways. Choosing network installation requires burning a single CD, which can be found in Debian web<sup>3</sup> or it can be directly [downloaded](#)<sup>4</sup>.
2. Switch on the computer connection to allow downloading of all required packages. A complete installation is advised, including KDE or Gnome.
3. If problems, check the [manual](#)<sup>5</sup>.

### 3.2 GCC 3.3.5 Installation

1. GCC 3.3.5 is required.
2. First, we check whether **apt-get** is well configured. We write: **apt-cache search gcc-3.3**. This should provide an output.
3. In no output was obtained
  - (a) Check whether **apt-get** is correctly installed.
  - (b) Check that the apt font directories are correctly located. Go to **/etc/apt/** directory and edit **sources.list** file with the following information.
    - deb <http://ftp.se.debian.org/debian/> stable main
    - deb-src <http://ftp.se.debian.org/debian/> stable main
    - deb <http://security.debian.org/debian/> stable/updates main
4. Install gcc **apt-get install gcc-3.3**. This will install the compiler for S.Ha.R.K. compilation. Debian may have other compilers. But this one seems to be required to compile the Kernel with zero errors.

---

<sup>2</sup>The estimated time is 4 to 5 hours

<sup>3</sup><http://www.debian.org>

<sup>4</sup><http://cdimage.debian.org/debian-cd/3.1.0a/i386/iso-cd/debian-31r0a-i386-netinst.iso>

<sup>5</sup><http://www.debian.org/releases/stable/i386/>

### 3.3 S.Ha.R.K. Installation

1. Installing any version of S.Ha.R.K. would suffice. We use version **1.5beta1** that was available [here](#)<sup>6</sup>
2. Our modified version can also be used.
3. Untar S.Ha.R.K., from the desktop or command line (**tar -xvzf shark-1.5beta1.tar.bz2**).
4. Go to the S.Ha.R.K directory and execute **make** and **make install**

### 3.4 Bochs Installation

1. Download bochs 2.2.1 for linux [here](#)<sup>7</sup>.
2. A [Windows](#)<sup>8</sup> version is also available. It comes compiled.
3. After untar Bochs from the Desktop or from the command line (**tar -xvzf bochs-2.2.1.tar.gz**) it has to be compiled. First **./configure**, **make** and **make install**.
4. Afterwards, Bochs has to be configured. Bochs upload its configuration from a plain text file that specifies which machine will emulate. For example:

```
romimage: file=./BIOS-bochs-latest, address=0xf0000
megs: 32
vgaromimage: file=./VGABIOS-lgpl-latest
4 floppy: 1_44=./shark.image, status=inserted
boot: a
ips: 1000000
floppy_bootsig_check: disabled=0
log: /dev/stdout
9 panic: action=ask
error: action=report
info: action=report
debug: action=ignore
debugger_log: -
14 com1: enabled=1, dev=/dev/ttyS0
vga_update_interval: 300000
keyboard_serial_delay: 250
keyboard_paste_delay: 100000
```

<sup>6</sup><http://shark.sssup.it/distrib/sources-1.5b1/shark-1.5beta1.tar.bz2>

<sup>7</sup><http://prdownloads.sourceforge.net/bochs/bochs-2.2.1.tar.gz?download>

<sup>8</sup><http://prdownloads.sourceforge.net/bochs/Bochs-2.2.1.exe?download>

```

19 floppy_command_delay: 500
   mouse: enabled=1
   private_colormap: enabled=0

```

- In line 1 we specify which **Bios** file is required for configuring the virtual PC.
  - In line 2 we specify the virtual PC memory.
  - In line 3 we specify the file that represents the configuration of the graphic card.
  - In line 4 we specify the image inserted in the virtual PC **a** unit.
  - In line 5 we specify the boot unit.
  - In line 6 we specify the clock frequency.
  - And so on. All lined have a meaning that can be checked in this [web](#)<sup>9</sup>.
5. From the directory where we have **Bochs**, we have to execute **bochs**. The first thing that will happen will be to look for the configuration file that we have in the current directory.

### 3.5 Booting S.Ha.R.K. HOWTO

Several ways of booting S.Ha.R.K. are described next:

1. **Direct Boot** We require a compiled example of S.Ha.R.K (**e.g.:** `./demos/jumpball/ecp`) and compile it into the `/boot` directory. After, we have to modify our boot loader (in our case GRUB, but with LILO should be similar). Open `/boot/grub/grub.lst` file and add the following lines in the corresponding location:

```

title S.Ha.R.K - jumpball ECP
root (hd0,0)
kernel /boot/ecp root=/dev/hda1 ro single
boot

```

2. **Boot from a floppy** We require a floppy able to boot. We have a floppy with GRUB installed and prepared to boot an image named `shark` located inside of the `/boot` directory. In order to copy an S.Ha.R.K. example in the floppy we proceed as follows::

<sup>9</sup><http://bochs.sourceforge.net/doc/docbook/user/bochsrc.html>

- mounting the floppy.
- `cp -f ecp /media/floppy/boot/shark` (o the corresponding floppy path).
- unmounting the floppy.

This floppy will be able to boot in any PC.

3. **Boot from Bochs** Taking advantage of having the previous floppy, and if we have configured **Bochs** as detailed before, when executing will go to the floppy and will boot it.

### 3.6 S.Ha.R.K for Dummies

- Install Debian and GCC (as explained above).
- Decompress all.tar file
- Insert a floppy in the floppy drive (floppy will be deleted).
- Go into `./All_in_one` directory and execute **make** and **make install**. This will compile **S.Ha.R.K**, **Bochs** and will create a bootable floppy with GRUB installed.
- Go to `./All_in_one/shark/demos/cs2/` directory and execute **make** (compile), **make install** (copy of the example into the floppy) and **./executa** (runs Bochs).

For performing modifications, look into the `./All_in_one/shark/demos/cs2/` directory how files **makefile** and **executa** look like.

### 3.7 S.Ha.R.K for even preDummies

Knoppix is a Linux distribution that can be made bootable from a CD. We have created a new Knoppix ISO, from the 4.0 version, where Bochs and S.Ha.R.K. have been added <sup>10</sup>.

All code can be found in `/usr/src` directory. In this directory we can find the following files or folders:

- bochs-2.2.1 Bochs fonts already compiled.
- shark S.Ha.R.K. fonts already compiled.

---

<sup>10</sup>The ISO can be found at <http://www.upcnet.es/~pmc16/knoppix.iso>

- sharkImage Floppy image to work with.
- sharkImageOrg Copy of the previous image for restoring it.

In `/usr/local/sbin` directory we will find a set of scripts that will help us working with S.Ha.R.K. This directory is included in the PATH. Therefore, we can work with these functions from anywhere. To work with them, (**root**) privileges are required. To become **root** in Knoppix we must type **su** from the command line.

- **Shark\_createDisket** It will create a bootable floppy with S.Ha.R.K. (the new image).
- **Shark\_loadDisket** It loads the floppy contents into the image.
- **Shark\_saveToDisket** Is saves the image in the floppy.
- **Shark\_mountImage** In mounts the image at `/home/sharkImage`
- **Shark\_umountImage** Unmounts the image.
- **Shark\_bootImage** Boots the image
- **Shark\_modifyImage** When compiling a Shark example, we obtain a "Shark file" (e.g.: if we type "make" in `/usr/src/shark/demos/jump-ball` directory, this will generate 4 executables `ecp`, `err`,...). If we want to place one of these files in the image in order to execute it, we have to do **Shark\_modifyImage nomDelFitxer** i when booting, this file will execute.
- **Shark\_restoreImage** It restores the image with the original content. This also happens if we reboot.

In `/etc` directory we have `bochssrc` file, which is the configuration file. Finally, in `/home` directory, we have `./sharkImage` directory, where we have the mounting point of the S.Ha.R.K. image (`mkdir /home/sharkImage`)

The structure of the working image is the following:

- `/boot`
  - grub Boot loader manager.
  - shark Image that will boot

- /config
  - config.mk S.Ha.R.K configuration file that allows compilation from the floppy.
  - exemple.mk As before.
- /demos
  - jumpball Example fonts.
    - \* Makefile Original file modified to permit running **make install** and copying the binary created at **/boot/shark** in order allow boot from floppy.

## 4 Conclusions

In this report we have described modifications made in the S.Ha.R.K. kernel to allow easy implementation of feedback scheduling policies. In addition, effort has been made on providing short but complete guidelines for installing all necessary software in order to allow quick utilization of the S.Ha.R.K. kernel.

## References

- [1] Soft and Hard Real-Time Kernel (S.Ha.R.K), <http://shark.sssup.it/>
- [2] P. Martí, C. Lin, S. A. Brandt, M. Velasco, J.M. Fuertes. “Optimal State Feedback Based Resource Allocation for Resource-Constrained Control Tasks,” in *Proc. 23rd IEEE Real-Time Systems Symposium*. 2004.
- [3] D. Henriksson, A. Cervin. “Optimal On-line Sampling Period Assignment for Real-Time Control Tasks Based on Plant State Information,” in *44th IEEE Conference on Decision and Control and European Control Conference ECC*. 2005.
- [4] Rosa Castañé, Pau Martí, Manel Velasco, Anton Cervin and Dan Henriksson. “Resource Management for Control Tasks Based on the Transient Dynamics of Closed-Loop Systems,” in *Proc. 18th Euromicro Conference on Real-Time Systems*. 2006.
- [5] Debian Linux, <http://www.debian.org/>
- [6] Bochs, <http://sourceforge.net/projects/bochs/>
- [7] Knoppix Linux, <http://www.knoppix.org/>