The S.Ha.R.K. Quick Guide

Tullio Facchinetti
tullio.facchinetti@unipv.it

Version 1.11

March 10, 2006

Contents

Chapter 1

Introduction

This document has been written for helping the beginner to start using the S.Ha.R.K. (**S**oft **HA**rd **R**eal-time **K**ernel) kernel. The S.Ha.R.K. kernel source code is available at

http://shark.sssup.it

The S.Ha.R.K. Quick Guide is intended to be as an ongoing document, since new topics will be added if needed.

1.1 What S.Ha.R.K. is

S.Ha.R.K. is a set of libraries that are statically linked into a multiboot image, that is the application. Then, the application is run through a DOS memory extender (X.EXE) or directly launched at boot time using the GRUB.

S.Ha.R.K. makes available to the application all the primitives to create, activate and run real-time tasks using state of the art scheduling algorithms (EDF, RM, etc.). It also allows to share data among the tasks by chosing between several resource reservation algorithms like Priority Ceiling, Priority Inheritance, SRP, etc. S.Ha.R.K. supports drivers for a huge amount of hardware, like video and network cards, frame-grabbers, multi-purpose cards and custom boards.

Shark is a compiling kernel with memory protection, not a kernel like Linux. You will not find any main() routine into the kernel tree because the tree only contains the set of libraries required to build the application. The main() function is called by the kernel, and is contained into the application files as specified by the C language standard.

Chapter 2

Getting started

This Section covers the S.Ha.R.K. installation and system building from the sources available on the website. As will be introduced in Section 2.1, S.Ha.R.K. can be used under Linux, Windows^{TM} or xDOS (preferably the free version of DOS, FreeDOS).

2.1 Supported platforms

S.Ha.R.K. application, as well as the Shark kernel, modules and drivers, can be developed under three main platforms: DOS (either MS-DOS^{TM} or FreeDOS[1]), Windows^{TM} or Linux. Table 2.1 shows some characteristics of using Shark under the three systems.

	DOS	Linux	Windows
Graphical interface	no	**yes**	**yes**
Good editors	it depends	**yes**	**yes**
Compilation speed	slow	**fast**	**fast**
Native compiler	no	**yes**	no
Programming/debug tools	no	**yes**	no
Dealing with long filenames	TSRs	**native**	**native**
Native execution	**yes**	no	no

Table 2.1: Quick comparison among the different supported platforms.

A quick consideration about the item "Good editors" for DOS. Someone may consider RHIDE, the DOS editor distributed with S.Ha.R.K. a good editor: I don't. But I also hate EDIT. Usually, I use an old editor called AURORA. More information at

http://www-personal.umich.edu/ knassen/aurora.html

Linux and Windows^{TM} are quite similar, in terms of programming facilities. The main advantage on using FreeDOS platforms is that it does not require

[1]In the rest of the document, we will only refer to FreeDOS while addressing the xDOS platform. Most of the topics hold for other DOS systems; specific indications for other platforms will be given if necessary.

to reboot the machine to execute the application, while when developing under Linux or Windows^{TM} it does.

Cygwin or other Linux console for Windows^{TM} are not currently and directly supported. However, the support to such platforms is not mandatory in the current version, since a Windows^{TM} user can easily build the system and develop application directly from the Windows^{TM} platform.

2.2 Installation

The installation changes a little bit from platform to platform.

2.2.1 Linux

1. download *shark-1.5.tar.bz2* from the S.Ha.R.K. web site (http://shark.sssup.it)
2. type *tar xvjf shark-1.5.tar.bz2*
3. type *cd shark*
4. edit the S.Ha.R.K. configuration file, *shark.cfg*
5. type *make all*
6. type *cd demos*
7. type *make*

Some tips:

- Step 4 is needed to setup the compiler options and optimize the kernel for speed and precision.
- Step 5 builds the S.Ha.R.K. kernel, libraries, modules and device drivers.
- Finally, step 7 builds the demo programs.

2.2.2 DOS

The installation for the xDOS platform follows:

1. download *unzip32.exe*, *mindj333.zip* and *shark15.zip* from the S.Ha.R.K. web site.
2. type *unzip32 -o mindj333.zip -d c:*
3. type *cd c:*
   djgpp
4. type *install.bat*
5. type *setvar.bat*
6. type *unzip32 -o shark15.zip -d c:*

7. type *cd c:*
   shark

8. edit *shark.cfg*

9. type *make*

10. type *cd demos*

11. type *make*

12. type *cd <demo dir>*

13. type *x <demo name>*

Some tips:

- If you already have *PkZip*, at step 2 you may also type *pkunzip -d -o mindj333.zip c:*

- If you change the DJGPP directory name or position, *install.bat* and *setvar.bat* need also to be changed.

- The *setvar.bat* script automatically set the environement variables for DJGPP; you must run *install.bat* and *setvar.bat* every time you reboot and start a compile session.

- After step 5, DJGPP is installed and ready to compile shark.

- If you already have PkZip, at step 6 you may also type *pkunzip -d -o shark15.zip c:*

- Step 8 is needed to setup the compiler options and optimize the kernel for speed and precision.

- Step 9 builds the S.Ha.R.K. kernel, libraries, modules and device drivers.

- Finally, step 11 builds the demo programs.

In DOS real mode we suggest, even though it is not mandatory, to load the smartdrive utility (*smartdrv 16000 /x*) to speedup the disk access during all the previous steps. The *smartdrv* utility is available for MS-DOS^{TM}. For the FreeDOS environemnt you can use the lbacache instead.

IMPORTANT: You need DOSLFN.ZIP (http://shark.sssup.it/distrib/doslfn.zip), a TSR to use long name files under DOS real mode. You have to run the doslfn command before starting to play with S.Ha.R.K. every time the machine is rebooted. You don't need it only if you are inside a DOS emulation window (see Windows Millennium/NT/2000/XP paragraph).

In real DOS environement, you can compile and run a demos without reboot, directly running the demo programs through the *X.EXE* extender from the FreeDOS command line.

2.2.3 Windows Millennium/NT/2000/XP

1. download unzip32.exe,mindj333.zip and shark15.zip from the S.Ha.R.K. web site;
2. go in a DOS emulation window (DOS prompt);
3. follow the previous procedure for normal DOS environment.

Reminder: YOU CAN NOT EXECUTE A S.Ha.R.K. APPLICATION THROUGH THE X.EXE EXTENDER WITHIN A DOS EMULATION WINDOW. So, to test a compiled demo programs you need to reboot with the FreeDOS bootdisk or something similar.

2.3 How to partition the hard-disk for using with S.Ha.R.K.

Even though it is often not needed and it may be unconfortable to setup an already partitioned machine, it may be very useful to setup a machine to host the S.Ha.R.K. kernel in school and university laboratories for teaching purposes.

This section is dedicated to the setup of an hard-disk for dedicating a partition to the installation of S.Ha.R.K. It will explain how to partition and format an hard-disk in order to install a multi-boot environment allowing to choose between three operating systems: FreeDOS and S.Ha.R.K., Linux and Windows^{TM}. The five basic steps to setup the installation are:

1. partition the hard disk;
2. install Windows^{TM};
3. install FreeDOS;
4. install Linux;
5. setup the boot loader.

The tools needed to perform the installation are:

- Linux fdisk utility;
- a copy of Linux installation CD;
- a copy of Windows^{TM} installation CD;
- some FreeDOS utilities;
- a copy of S.Ha.R.K.

To accomplish all the operation related to the hard-disk partition, I suggest to use a Linux LiveCD distribution. My favourite one is Knoppix, by Mark Knopper, downloadable from

http://www.knoppix.org.

Linux LiveCD distribution allow to use all the powerful Linux tools without the need of installing anything on the hard-disk.

Boot the PC from the CD drive after inserting the Knoppix LiveCD into the drive and open a shell. You must be the root user to have the permissions for the next steps, so type

```
su
```

with an empty password at the console prompt. Then launch the Linux *fdisk*, typing

```
fdisk /dev/hdXXX
```

where XXX identifies your hard-drive. Typically, the command

```
fdisk /dev/hda
```

will work fine. To use the *fdisk* commands do refer to the program help: you will need the commands to delete and create partition as well as to change the partition ID and write the partition table on the disk.

I suggest to partition your HD like something as described in Table 2.2.

	Partition number	Dedicated to	Type
primary partitions	1st	FreeDOS	FAT32
	2nd	Windows	FAT32/NTFS
	3rd	Linux	extX/Reiser/...
extended partition	4th	Data 1	any
	...	...	...
	last but one	Data n	any
	last	Linux swap	(if needed)

Table 2.2: Example of hard-disk partitioning.

IMPORTANT: The FreeDOS partition must be the FIRST partition on your disk or you should experience some problems while booting FreeDOS.

Since *fdisk* creates all the partitions as Linux partitions, you must toggle the partition type of the 2nd partition to make it a FAT32 (LBA) or NTFS.

SUGGESTION: Do leave all the other partition as Linux type, so that Windows^{TM} will put all the booting stuff into the only partition that it can see at installation time.

Now, reboot and install Windows^{TM} into the only partition that is available for it. I only tested the 2000 and XP versions of Windows. If you want to install previous Windows^{TM} versions (98 or 95) I'm not sure that they will be able to boot if they are not installed into the first partition. However if you want to use them, you can skip the installation of FreeDOS, since you can use the MS-DOS shell to run Shark programs.

After the installation of Windows^{TM}, boot Knoppix again and use *fdisk* to change the partition type of the 1st partition: toggle it as FAT32 (LBA) - suggested - or FAT16 if you want to test the filesystem capabilities of Shark.

Do format the 1st partition using the Windows formatting program.

Install FreeDOS. You can download all the packages from http://www.freedos.org.

I suggest to download the packages and to copy them manually into the partition, avoiding the Installation CD or floppy. You can also use a S.Ha.R.K. boot CD to copy the whole FreeDOS tree onto the disk.

Make a FreeDOS boot floppy using the *sys* command, executing a command like

sys a:

Copy also the *sys.com* FreeDOS program on the floppy.

Reboot the machine leaving the floppy into the drive and launch the command

sys x: x:
fdosboot.img

ATTENTION: "x:" refers to the drive letter of the partion in which FreeDOS has been installed, as seen by FreeDOS after the boot from floppy (check which is the right drive letter by listing the content of the drives with "dir x:").

What you did is to created an image of the boot sector which is able to boot FreeDOS, but instead of overwriting the Master Boot Record (MBR), you writed it onto the disk.

Now insert the Linux intallation disk, reboot the PC and install Linux. After the installation, the GRUB boot manager should be installed in your MBR (I did not tested other boot managers, like LILO).

Look for the configuration file of the GRUB, *grub.conf*, that is often placed into the */boot/grub* directory. The lines to boot Linux are usually added automatically by the installer. The lines to boot Windows^{TM} look like the following

title Win2000 rootnoverify (hd0,1) chainloader +1

Then, add the lines to boot FreeDOS:

title FreeDOS - Shark rootnoverify (hd0,0) makeactive chainloader /fdosboot.img boot

Now you are ready to install the S.Ha.R.K. distribution following the instructions of Section 2.2.

2.4 Kernel and applications

S.Ha.R.K. currently supports the PC platform (Intel^{TM} or AMD^{TM} processors, etc.). Using the S.Ha.R.K. kernel is easy. Few tools are required to build the kernel, to develop new applications and to run existing or new applications.

2.4.1 Building kernel and applications

S.Ha.R.K. is written in C, so it requires a C compiler to build the system from the source code. Currently, S.Ha.R.K. supports the *GCC version 3.3*, which is the standard compiler and is already present in most of the Linux distributions.

ATTENTION: the *GCC version 3.4* does not properly work while building the system or compiling user programs. You can check your GCC version by typing *gcc -v*

into a Linux console.

Windows^{TM} or FreeDOS come without their own C compiler. There exist a version of the GCC which runs under FreeDOS, and Windows^{TM} too, called DJGPP and available at

http://www.delorie.com/djgpp.

However, the DJGPP version downloadable from the DJ Delorie website it is not suitable to be used with S.Ha.R.K. since it produces the COFF[2] executable file format only. S.Ha.R.K. needs a compiler which has to be able to produce the ELF file format. This is why the DJGPP has been rebuilt in order to produce the ELF format.

ATTENTION: you must download the correct DJGPP version from the S.Ha.R.K. website.

2.4.2 Executing applications

There are two options to run a new application or one of the demos distributed with S.Ha.R.K.:

- using the X.EXE DOS memory extender;
- loading the application from the GRUB boot manager.

Using X.EXE, which is available on the S.Ha.R.K. website, requires an xDOS platform. Supported platforms are FreeDOS and MS-DOS^{TM}. In the latter case, however, an application can be run using X.EXE only from a Windows^{TM} 95, Windows^{TM} 98 and Windows^{TM} 95SE DOS console.

ATTENTION: Windows^{TM} Millennium/ NT/2000/ XP consoles can not be used to run S.Ha.R.K. applications.

Using the GRUB is more complicated and, since it is particularly useful for the remote execution of S.Ha.R.K. programs, this topic will be covered in a dedicated section.

2.4.3 Getting a DOS environment for executing S.Ha.R.K. applications

To execute a S.Ha.R.K. application a DOS environment is required, and several options are available to get it:

- an hard-disk can be partitioned and formatted for multi-boot, and a partition can be reserved to FreeDOS. This may be a tricky way, and it is suggested only if the users has a good knowledge of the involved operations. It is particularly useful to setup the workstations in school and university laboratories.

[2]See http://www.theparticle.com/cs/bc/os/elfpecoff.html for some information on executable file formats.

- re-booting a machine every time an execution of a S.Ha.R.K. application is needed with the S.Ha.R.K. boot CD or the boor floppy available on the S.Ha.R.K. website.

  The second option needs some more indications. When the machine restarts, a FreeDOS environment is made available. This means that it is possible to launch the X.EXE extender to run a S.Ha.R.K. application. The application itself should be put on the floppy or, anyway, it MUST be put on a partition with a filesystem reachable by the FreeDOS kernel. At the best of my knowledge, FreeDOS currently supports FAT16 and FAT32 filesystems only. So that, an application which is stored that resides on a NTFS (Windows^{TM}), ext2, ext3, ReiserFS or other Linux filesystems can not be executed using the boot from floppy or CD.

Chapter 3

Using S.Ha.R.K.

3.1 Your first S.Ha.R.K. application

A typical S.Ha.R.K. application is composed by 3 main components: the *program* source file, the initialization source file (*initfile*) and the *makefile*.

3.1.1 The makefile

The makefile is the script used by the *make* command to build the application. Typically, the makefile is named "makefile", and you can build the application by simply launching the command *make* from the command line. However, the makefile may be named differently in order, for example, to maintain different make settings into the same directory. With a makefile named "myapp.mak", the application can be built typing the command "*make -f myapp.mak*".

IMPORTANT: To build an application, you can not launch the compiler directly without using the makefile, since the makefile sets several variables and calls several sub-routines before invoking the compiler.

The makefile contains some directives to specify the application name, the source file names and the libraries needed for building the application. Figure 3.1 shows an example of makefile for building an application called *myapp*, thus contained into the source file *myapp.c*. The application code is also contained into two source files: *source1.c*, *source2.c*. For each of those files, the compiler

```
ifndef BASE
BASE=../..
endif
include $(BASE)/config/config.mk
PROGS = myapp
include $(BASE)/config/example.mk
myapp:
   make -f $(SUBMAKE) APP=myapp INIT= OTHEROBJS="initfile.o
source2.o source3.o" OTHERINCL= SHARKOPT="__LINUXC26__
__PCI__ __INPUT__ __FB__"
```

Figure 3.1: Example of makefile.

produces an *object file*, with the ".o" extension. The object files required for making the application must be specified with the OTHEROBJS directive into the instruction at line 8. Notice that, since also the initfile contains the application source code (Section 3.1.2), you must include the related object file name into the OTHEROBJS directive.

To customize the makefile in order to build your own application, you must supply the program name at lines 5 and 8, and the same name should be used for the instruction at line 7.

At line 2, you specify the path where the S.Ha.R.K. kernel has been installed. Althought you can specify an absolute path like "/home/username/shark" for Linux or "c:\shark", typically a relative path is used. In Figure 3.1, the instruction at line 2 says that the base S.Ha.R.K. directory is located two levels before the application directory. For example, if S.Ha.R.K. has been installed into

/home/toolleeo/shark,

the application stays into

/home/toolleeo/shark/projects/apps.

The BASE directory in the example is relative to the current application directory. This works fine if the application directory is under the S.Ha.R.K. derictory tree. If the application stays outside the system tree, you can use an absolute indication of the S.Ha.R.K. base directory, such as "/home/toolleeo/shark" instead of "../..".

Finally, the instruction at line 8 specifies also the libraries that must be linked together the object code to build the whole application. In the example, the application uses the Linux 2.6 compatibility layer, which is always required when the drivers are used, the PCI driver, the Input driver (to manage the input devices like keyboard, mouse, etc.) and the Frame Buffer driver for the graphical display. The list of all the available libraries can be found into the *$(BASE)/lib* directory, where *$(BASE)* is the S.Ha.R.K. installation directory.

ATTENTION: when modifying the makefile, be sure to use a text editor which saves the TAB character and avoids editors (often you can configure this option into the editor preferences) which substitute TAB characters with spaces. This is important because instruction 8 MUST be indented with a TAB character. Otherwise, the *make* command will not be able to interpret the makefile correctly.

3.1.2 The initfile

The inifile is a normal source file which contains the instruction for the program initialization. The initfile includes the system header, initializes the modules regarding the scheduling policies for tasks and shared resources, the graphical mode if needed, the keyboard and all the required devices.

When the application is launched, before starting the multitasking mode the program must initialize the devices, the resources and the schedulers which will be used by the application. For doing so, the kernel calls the "__kernel_register_levels__" function, that usually registers the following modules (see the S.Ha.R.K. Kernel architecture Manual for more details):

- scheduling modules: a scheduling module implements a particular scheduling algorithm, for example EDF, RM, Round Robin, etc.;
- resource Modules: a resource module implements a shared resource access protocol (for example the semaphores, the mutexes, etc.);
- other devices, such for example the file system, and other devices that has to be initialized before entering the multitasking mode.

3.1.3 The program

The program file is the source file containing the *main* function

int main(int argc, char **argv)

The main function is automatically called by the kernel when the application is launched through the X memory extender or the GRUB. The main functions contains the declarations for - or calls the routine to initialize - the user tasks, the keyboard handler and all the components required by the application.

3.2 Tips and tricks

3.2.1 The BASE directory

S.Ha.R.K. uses the BASE variable to identify the directory where the system directory tree starts. For example, a typical installation under Linux may be under "/home/username/shark"; under FreeDOS the path would be "c:\shark".

Unfortunately, the OSLib component of S.Ha.R.K., which is developed separately from S.Ha.R.K., also requires to set a variable named BASE when building. Recalling the previous examples, the BASE values for the OSLib will be "/home/username/shark/oslib" and "c:\shark\oslib".

A problem during the kernel compilation arises if the BASE variable is set into the OS environment[1].

IMPORTANT: when compiling S.Ha.R.K., be sure that BASE is not set environment variable to get no problem[2].

During the application development, thus when S.Ha.R.K. is already built and you do not need to rebuild the kernel, you may set BASE to point to the S.Ha.R.K. base directory. In this sistuation, you can move the application source code around your filesystem without changing the BASE definition into the makefile. See Section 3.1.1 for an overview on the S.Ha.R.K. makefiles. However, you should avoid this practice; you should prefer to set correctly the BASE directory into the makefile for each project you are working at.

3.2.2 Text output

There are few things to know for using the S.Ha.R.K.'s functions to display text. Four functions should be used for a safe text output management:

[1]At the FreeDOS command line, type "*set*" to get the content of the environment.

[2]At the FreeDOS command line, type "*set BASE=*" to remove an already set BASE variable.

- *sprintf()* to write a formatted string into a generic text buffer (ex., an array of chars);

- *cprintf()* for the regular text output while the application is in text mode;
- *grx_text()* for the text output when the application is in graphical mode (to be used together with sprintf(), which generates the string to be displayed);
- *printk()* for displaying strings in text mode into the kernel and modules for debug purposes.

3.2.3 I/O to the filesystems

S.Ha.R.K. currently supports only FAT16 filesystems on IDE drivers. This means that, while the kernel is running, only FAT16 filesystems could be accessed from the application.

However, typically, read/write operations to the filesystem can be done at the beginning or at the end of the application, to load initial parameters or data and to save the results (which could be contained in a memory buffer filled during the application execution). To do so, do use the functions listed into *include/ll/i386/x-dos.h*, and allow the user to read and write binary and text files.

The difference between the full support of FAT16 filesystems and the functions of the *x-dos* library is that, while the former is a true driver which accesses the hard-disk hardware, the latter are hooks to DOS INT 21h functions. Since DOS INT 21h are not reentrant (basically, they can not be interrupted by the kernel), they can not be used while real-time tasks are running.

3.2.4 DJGPP and DOS utils

The DJGPP, the compiler used by S.Ha.R.K. under DOS, does not include any replacement for some Unix commands required to build S.Ha.R.K., like *cp*, *rm*, *mv*, etc. Those comands can be found easily in the Internet, but it is safer to use the ones included into the S.Ha.R.K. package.

Someone may already have its own set of command replacements. This may produce sevral problems, since different implementations manages differently the special command options. This is why S.Ha.R.K. is guaranteed to correctly build with the tested commands only (the Linux versions are all right).

Short anecdote[3]. The first version of the S.Ha.R.K. bootable CD included the correct commands got from the S.Ha.R.K. distribution package. It also included another version that is put into the "\fdos\bin" directory. Both the versions of the commands were reacheable through the PATH environment variable. However, for a mistake, the FreeDOS path was inserted into the PATH variable before the correct S.Ha.R.K. path, making FreeDOS using the wrong commands. Results: the makefiles were invoking the wrong command versions, producing several problems while building kernel and applications.

[3]I add this anecdote since it may help in solving related problems.

3.3 From version 1.4 to 1.5

From the user point of view, few things are changed from the S.Ha.R.K. 1.4 version to the current 1.5 version. The changes that make an application developed for version 1.4 incompatible with the 1.5 version are only two:

1. to make the new version compatible with the POSIX standard, functions *sys_end()* and *sys_abort()* have been replaced by the *exit(.)* function, which accepts an integer exit code parameter.

2. the S.Ha.R.K. directory tree has been slightly changed.

The main effect of the changes to the tree are that an application initfile now requires slightly different instructions to include the needed modules. A comparison of the instructions required for the two S.Ha.R.K. versions is depicted in Table 3.1.

Version 1.4	Version 1.5
#include "kernel/kern.h"	#include "kernel/kern.h"
#include "modules/intdrive.h"	#include "intdrive/intdrive/intdrive.h"
#include "modules/edf.h"	#include "edf/edf/edf.h"
#include "modules/cbs.h"	#include "cbs/cbs/cbs.h"
#include "modules/rr.h"	#include "rr/rr/rr.h"
#include "modules/dummy.h"	#include "dummy/dummy/dummy.h"
#include "modules/sem.h"	#include "sem/sem/sem.h"
#include "modules/hartport.h"	#include "hartport/hartport/hartport.h"
#include "modules/cabs.h"	#include "cabs/cabs/cabs.h"
#include "modules/pi.h"	#include "pi/pi/pi.h"

Table 3.1: The differences between inifiles in S.Ha.R.K. 1.4 and 1.5.

3.4 Remote execution of S.Ha.R.K. applications

S.Ha.R.K. supports the Multiboot[4] standard. This means that it is possible to launch remotely a S.Ha.R.K. application from, for example, the GRUB[5] boot loader.

Since this Section will describe how to boot a S.Ha.R.K. application using the GRUB, let's say something more about it, and boot loaders in general.

In very simple words, boot loaders are programs that, taking the control of the machine after the system start-up, launch the operative system kernels. Boot loaders are particularly useful because they let you choose between different operative systems to start. Section 2.3 illustrated how to partition an hard-disk to install more than one OS. This Section will describe how to load a kernel (in our case, a S.Ha.R.K. application) from the network and run it on a remote machine.

An example will explain the installation procedure and how to use this mechanism.

[4]See http://www.gnu.org/software/grub/manual/multiboot/ for more information.

[5]See http://www.gnu.org/software/grub/ for more information.

3.4.1 General setup

Suppose that there are two machines: computer A, where you develop your S.Ha.R.K. applications using the Linux platform, and computer B, where you want to execute the application.

Computer A needs:

- an Ethernet connection with B;
- Linux;
- an installed and properly configured version of S.Ha.R.K.;
- a TFTP server.

Computer B needs:

- an Ethernet connection with A;
- an installation of the GRUB with network support.

3.4.2 Computer A setup

There are two important things to setup on machine A. First of all, the S.Ha.R.K. kernel must be compiled with the option BIOS set to VM86 (Virtual Memory for x86) into the shark.cfg configuration file. This ensures that the application will be compiled with the support for the multiboot.

ATTENTION: as stated in the configuration file, the VM86 mode is not compatible with some VGA cards (like Matrox).

A TFTP server must be installed on computer A, to let B download the application. TFTP means Trivial File Transfer Protocol; it is a simplified version of FTP which permits only basic operations like download and upload (no directory management, etc.). Moreover, it works over the UDP protocol, while FTP works over TCP. Almost all Linux distribution have a TFTP server. I will not discuss how to install and run it. The only important thing to know is that, typically, the server lets a remote machine to download a program from a specific directory (e.g., /tftpboot/). The compiled S.Ha.R.K. program will be put into that directory.

3.4.3 Computer B setup

First of all, let's build the GRUB for supporting the network. Currently, the GRUB 2 is under development. I only tested the remote execution with S.Ha.R.K. using the so called the GRUB Legacy, which is still available.

Download the GRUB Legacy from

ftp://alpha.gnu.org/gnu/grub/

Since I only tested the behavior of the *grub-0.97.tar.gz* package, I suggest to use this version.

After unpacking the sources on a Linux machine with

tar -xvzf grub-0.97.tar.gz

you will get the *grub-0.97* directory. Do get into it using

cd grub-0.97

and configure the GRUB with a command like

./configure –enable-via-rhine –disable-ffs –disable-minix –disable-reiserfs –disable-xfs –disable-ufs2 –disable-vstafs

This instruction simply configures the GRUB to support a VIA network board (Rhine model) and turned off the support for many filesystems, like FFS, Minix, XFS and others.

Of course, if you have a different network board, you need to enable the correct one. You can find a list of the supported network boards under the directory *grub-0.97/netboot*, into the *README.netboot* file. The GRUB developers sugget to not enable too many network cards all together for stability problems[6]. For the same reason, all the unused filesystems have been disabled. The above configuration assume that the GRUB files stay into a directory called */boot/grub*, which is into an ext2/ext3 filesystem. If your */boot/grub* directory on a filesystem of different type, you must enable the support for that specific filesystem.

After the GRUB configuration has been correctly performed, type

make

to launch the build of the GRUB.

After the compilation, you will find two files, *stage1* and *stage2*, into two separate directories with the same name. You must copy the two files into the */boot/grub* directory of computer B. Before copying the files, do make a backup copy of the existing textitstage1 and *stage2*. This step assumes that the GRUB is already installed on computer B, which is typically done when you install Linux on a PC: we only want to enhance the GRUB to support the network boot.

Then, add the following lines to the *menu.lst* script file into the */boot/grub* directory:

- `ifconfig --address=192.168.0.2 --mask=255.255.255.0`
- `tftpserver 192.168.0.1`
- `rootnoverify (nd)`
- `kernel /shark`
- `boot`

Where 192.168.0.2 is the IP address assigned to computer B and 192.168.0.1 is the IP address of computer A. Change those values to accomplish with the need of your local network.

Finally, reboot the machine and, at the GRUB prompt, type "*c*" to get the commad line of the GRUB, and type the command

[6]This may be the reason because it is not safe to build a copy of the GRUB with the native support of all the available network cards.

install (hd0,2)/boot/grub/stage1 (hd0) (hd0,2)/boot/grub/stage2 0x8000 (hd0,2)/boot/grub/menu.lst

This instruction install the GRUB into the Master Boot Record of your hard-disk. Typing *(hd0,2)* you specify that the */boot/grub/* directory stays into the 3rd partition of your first hard-disk. You must change this value if the directory is into another partition or another disk[7].

3.4.4 Executing S.Ha.R.K. application

After the setup of the GRUB and the TFTP server, you are able to run a S.Ha.R.K. application by compiling the application and copying it into the TFTP base directory (e.g., */tftpboot*) and calling it *shark*. This is required because, after the configuration of *menu.lst* described in Section **??**, the GRUB will look for a Multiboot kernel named *shark*.

Every time you reboot computer B, it will automatically look for the *shark* multiboot image to run it. To speed up the GRUB operations, you should set the remote S.Ha.R.K. application execution as the default option and, optionally, do set a low number of seconds for the GRUB initial waiting time.

There are some drawbacks of the previous installation. First, the S.Ha.R.K. application name must be *shark*; you may change that name, however the point is that you must change the *menu.lst* to use different names. Second, the S.Ha.R.K. application can not use the filesystem to read and write data, so do not put any I/O command to file into an application remotely executed. Finally, the S.Ha.R.K. application can not accept command line parameters.

3.4.5 Accepting command line parameters

To make a remotely executed S.Ha.R.K. application to accept command line parameters you can exploit a feature of the GRUB which allows to remotely load a configuration file from the TFTP server.

To obtain this result, put the following instruction into the *menu.lst* file on computer B:

- `ifconfig --address=192.168.0.2 --mask=255.255.255.0`
- `tftpserver 192.168.0.1`
- `rootnoverify (nd)`
- `configfile /menu-remote.lst`

The last instruction tells to the GRUB to download the *menu-remote.lst* configuration file from the TFTP server.

Put a file named *menu-remote.lst* into the TFTP base directory of computer A (*/tftpboot/*). Put the following instructions unto the *menu-remote.lst*:

- `kernel /shark-app param1 param2 param3`

[7]While *hd1* identifies the 2nd disk, *hd2* the 3rd, etc., *(hd0,0)* is the 1st partition of the 1st disk, *(hd0,1)* is the 2nd partition of the 1st disk and so on. This is a convention of the GRUB.

- `boot`

In this way, by editing the *menu-remote.lst* on the machine where you are developing your S.Ha.R.K. applications, you are able to change the name of the S.Ha.R.K. application that will be looked for by the GRUB. Moreover, you can use as many command line parameters as needed when the S.Ha.R.K. application starts.