Hello MaRTE OS using an emulator

Daniel Sangorrin

daniel.sangorrin@unican.es

2009-3-4

Revision History

Revision 1.3 2009-3-4 Revised by: dsl
Add mtools for mounting the images. Update installation instructions referencing INSTALL file.
Revision 1.2 2007-6-15 Revised by: dsl
Added the process to create a FAT16 image file for QEMU
Revision 1.1 2007-6-9 Revised by: dsl
Updated to MaRTE OS v1.6 snapshot

This document tries to introduce people to MaRTE OS in a very simple way.

Table of Contents

1. Preface

This document is intented for those who have read MaRTE OS site and are curious about how a MaRTE OS applicaton looks like or how difficult or easy is to get it working. Other documentation sources of your interest might be the user's guide and the INSTALL file, both included in standard MaRTE OS distributions.

1.1. Feedback

Feedback is most certainly welcome for this document. Send your additions, comments and criticisms to the following email address : <daniel.sangorrin@unican.es>.

2. Let's see MaRTE OS running!

I know you are feeling like watching some DEMO to see how a MaRTE OS application looks like. Don't worry, you'll get it done in no time!

2.1. Install the emulator QEMU...

Instead of running MaRTE OS on a physical bare x86 machine we are going to use an x86 emulator. An emulator is very useful because you don't need another machine for testing your application (you can also test it on the host but you would need to reset all the time). We have chosen a great GNU/GPL emulator called QEMU.

QEMU is very simple to install. For the binary distribution all you need to do is **untar** it on `'/'`. You can also get the sources and compile them (the best if you want to have the latest version or you need to compile it with specific flags like debugging).

These are the instructions for installing the binary distribution. You can get it at QEMU site [1]. Note that the commands are executed as **root**. If you can't be **root** then try compiling the sources using the option **--prefix=/your-path**

```
[root@machine: dir]# cd /
[root@machine: dir]# tar zxvf qemu-x.x.x-i386.tar.gz
```

2.2. ...and run these MaRTE OS Demos!

Files bellow are disk images that you can boot with QEMU. Each one have a MaRTE OS application demo ready to run.

MaRTE OS Demos:

1. hello_marte.img.tar.gz: This demo application simply put a message on the screen and ends.

   ```
   $ tar zxvf hello_marte.img.tar.gz
   $ qemu -fda hello_marte.img
   ```

2. music_demo.img.tar.gz: This demo application plays a song. The song title is "OPEN SOURCE" and is copyrighted (c) 2006 by Magic Mushrooms and distributed under LinuxTag Green OpenMusic License terms (http://openmusic.linuxtag.org/)

```
$ tar zxvf music_demo.img.tar.gz
$ qemu -hda music_demo.img -soundhw sb16
```

They consist of a GRUB boot loader and a demo application developed with MaRTE OS (**mprogram**). Their structure looks like this:

```
demodisk/
   |-- boot
   |   `-- grub
   |       |-- device.map
   |       |-- fat_stage1_5
   |       |-- menu.lst
   |       |-- stage1
   |       `-- stage2
   `-- mprogram
```

3. Hello MaRTE!

Don't be afraid anymore, MaRTE OS is very simple to install and use!. Like when you're learning a new language we are going to start with a simple *hello world* example (in this case a *hello marte*).

3.1. The compiler GNAT-GPL

MaRTE OS is written in Ada, then even in the case you are not going to program Ada applications, you will need the GNAT compiler to compile the MaRTE kernel. Installing it is very easy and can be done in your own user directory without affecting the rest of the system. Installing AdaCore GNAT-GPL compiler is very simple. Just go to its web-site at https://libre.adacore.com/ and, after registering, download the compiler. Then, you have to decompress it and execute a simple script. More detailed instructions can be found in MaRTE OS INSTALL file.

3.2. MaRTE OS

All we need now is to download MaRTE OS sources, install them and compile an application. The detailed instructions can be found in MaRTE OS INSTALL file.

After compiling the GNAT runtime and the MaRTE OS kernel, it will be ready to be linked with our application. Open `/home/user/marte/examples/hello_world.adb` or `/home/user/marte/examples/hello_world_c.c`, change something if you want and compile it:

```
$ mgnatmake hello_world.adb
$ mgcc hello_world_c.c
```

If everything went ok we should have our application linked to the MaRTE OS kernel into a file named **mprogram** at `/home/user/export`.

Now we need to *boot* this application. Download this floppy image (hello_marte.img.tar.gz) disk with GRUB boot loader already installed, and perform the following operations:

```
$ tar zxvf hello_marte.img.tar.gz
$ mkdir myfloppy
$ mount -o loop hello_marte.img myfloppy
$ cp mprogram myfloppy/
$ umount myfloppy/
```

Time to test it with QEMU!

```
$ qemu -fda hello_marte.img
```

4. Creating IMG files for QEMU

In the previous example we used a floppy image file with GRUB but I didn't explain how I created it. Furthermore, it would be nice to have a normal hard disk image file so we are not limited in size. In this section we will see how to create this file from scratch and how to use QEMU to emulate our mprogram's with it. NOTE: you can download the QEMU images already created from MaRTE OS Website.

The overall idea is that we are going to create a file full of zeros, create a partition in it, format that partition with a FAT16 filesystem and install the GRUB bootloader in it. Now, let's go to the point! (Note: I am using Ubuntu Dapper for this, I hope it works in other distributions as well). I will put the command first and then a explanation.

```
$ sudo -s
```

We need superuser rights. For other distributions you might have to use the **su** command instead of **sudo**.

```
# dd if=/dev/zero of=disk.img bs=516096c count=20
```

We create a zeroed image file. Here you can modify the **count** parameter in order to make your image file bigger or smaller. **count** denotes the number of cylinders, and the formula that relates it to the size of the file is: size=count*16*63*512, where 512 comes from the number of bytes of each disk sector.

```
# losetup /dev/loop/0 disk.img
```

The losetup command (http://www.linuxcommand.org/man_pages/losetup8.html) associates our image file to a loop device.

```
# fdisk -u -C20 -S63 -H16 /dev/loop0
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklab el
Building a new DOS disklabel. Changes will remain in memory only,
until you decide to write them. After that, of course, the previous
content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help): o
Building a new DOS disklabel. Changes will remain in memory only,
until you decide to write them. After that, of course, the previous
content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help): n
Command action
e   extended
p   primary partition (1-4)
p
Partition number (1-4): 1
First sector (63-20159, default 63):
Using default value 63
Last sector or +size or +sizeM or +sizeK (63-20159, default 20159):
Using default value 20159

Command (m for help): t
Selected partition 1
Hex code (type L to list codes): L

0  Empty           1e  Hidden W95 FAT1 80  Old Minix       be  Solaris boot
1  FAT12           24  NEC DOS         81  Minix / old Lin bf  Solaris
2  XENIX root      39  Plan 9          82  Linux swap / So c1  DRDOS/sec (FAT-
3  XENIX usr       3c  PartitionMagic  83  Linux           c4  DRDOS/sec (FAT-
4  FAT16 <32M      40  Venix 80286     84  OS/2 hidden C:  c6  DRDOS/sec (FAT-
5  Extended        41  PPC PReP Boot   85  Linux extended  c7  Syrinx
6  FAT16           42  SFS             86  NTFS volume set da  Non-FS data
7  HPFS/NTFS       4d  QNX4.x          87  NTFS volume set db  CP/M / CTOS / .
8  AIX             4e  QNX4.x 2nd part 88  Linux plaintext de  Dell Utility
9  AIX bootable    4f  QNX4.x 3rd part 8e  Linux LVM       df  BootIt
a  OS/2 Boot Manag 50  OnTrack DM      93  Amoeba          e1  DOS access
b  W95 FAT32       51  OnTrack DM6 Aux 94  Amoeba BBT      e3  DOS R/O
c  W95 FAT32 (LBA) 52  CP/M            9f  BSD/OS          e4  SpeedStor
e  W95 FAT16 (LBA) 53  OnTrack DM6 Aux a0  IBM Thinkpad hi eb  BeOS fs
f  W95 Ext'd (LBA) 54  OnTrackDM6      a5  FreeBSD         ee  EFI GPT
10  OPUS            55  EZ-Drive        a6  OpenBSD         ef  EFI (FAT-12/16/
11  Hidden FAT12    56  Golden Bow      a7  NeXTSTEP        f0  Linux/PA-RISC b
12  Compaq diagnost 5c  Priam Edisk     a8  Darwin UFS      f1  SpeedStor
14  Hidden FAT16 <3 61  SpeedStor       a9  NetBSD          f4  SpeedStor
16  Hidden FAT16    63  GNU HURD or Sys ab  Darwin boot     f2  DOS secondary
```

```
17  Hidden HPFS/NTF 64  Novell Netware  b7  BSDI fs         fd  Linux raid auto
18  AST SmartSleep  65  Novell Netware  b8  BSDI swap       fe  LANstep
1b  Hidden W95 FAT3 70  DiskSecure Mult bb  Boot Wizard hid ff  BBT
1c  Hidden W95 FAT3 75  PC/IX
Hex code (type L to list codes): 6
Changed system type of partition 1 to 6 (FAT16)

Command (m for help): a
Partition number (1-4): 1

Command (m for help): p

Disk /dev/loop0: 10 MB, 10321920 bytes
16 heads, 63 sectors/track, 20 cylinders, total 20160 sectors
Units = sectors of 1 * 512 = 512 bytes

Device Boot      Start         End      Blocks   Id  System
/dev/loop0p1   *          63       20159       10048+   6  FAT16

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error 22: Invalid argument.
The kernel still uses the old table.
The new table will be used at the next reboot.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.
```

Using the tool **fdisk** we create a primary partition starting after the sector 63, we set its type as FAT16 so the GRUB bootloader knows what filesystem module use and the bootable flag to 1.

```
# losetup -d /dev/loop0
# losetup -o32256 /dev/loop0 disk.img
```

We detach the file associated with the specified loop device using **losetup**, and reassociate it with the offset of the first sector of the partition (63*512=32256 bytes)

```
# mkdosfs -F16 /dev/loop0 10048
mkdosfs 2.11 (12 Mar 2005)
Loop device does not match a floppy size, using default hd params
```

Using the **mkdosfs** tool we format the partition with a FAT16 filesystem. The argument 10048 is the number of blocks and it can be taken from the output of the previous **fdisk** (as you can see if you look above)

```
# mkdir mydisk
# mount /dev/loop0 mydisk/
# mkdir -p mydisk/boot/grub
# cp /boot/grub/* mydisk/boot/grub/
# cp (your-marte-path)/mprogram mydisk/
# vi mydisk/boot/grub/menu.lst
# cat mydisk/boot/grub/menu.lst
default         0
timeout         3
title           MaRTE OS
root            (hd0,0)
kernel          /mprogram
```

Ok, now we have a disk image file with a FAT16 filesystem. We mount it in a directory and copy inside all the necessary GRUB bootfiles (I take them directly from Ubuntu). We modify the GRUB configuration file so it will boot our **mprogram**.

```
# umount mydisk/
# grub --no-floppy
>> device (hd0) disk.img
>> root (hd0,0)
>> setup (hd0)
>> quit
# losetup -d /dev/loop0
```

After umounting our disk we install GRUB by executing **grub** and giving him the above parameters.

```
$ qemu -hda disk.img
```

Finally, we execute QEMU with our new image file as the hard disk.

Now, in order to change the **mprogram** inside the disk image file we can use the following commands (You can have them in a file with execute permissions and use it as a script instead of writing all the commands) as root.

```
losetup -o32256 /dev/loop/0 disk.img &&
mount -o loop /dev/loop0  mfloppy &&
cp -f mprogram mfloppy/mprogram &&
umount mfloppy &&
losetup -d /dev/loop/0
```

Another option that does not require root mode and is simpler is to install the 'mtools' (i.e.: sudo apt-get install mtools) and put a configuration file called `.mtoolsrc` in the home directory:

```
drive c:
file="/home/user/disk.img"
partition=1
```

```
mtools_skip_check=1
```

Finally, copy the mprogram into the image file:

```
mcopy -o mprogram c:
```

Or just use the script in MaRTE OS utils

```
mqemu mprogram
```

5. Further Information

1. QEMU http://fabrice.bellard.free.fr/qemu/download.html (http://fabrice.bellard.free.fr/qemu/download.html)
2. GNAT-GPL https://libre2.adacore.com/ (https://libre2.adacore.com/)
3. MaRTE OS User's guide and INSTALL (inside MaRTE OS distribution)

6. Legal section

6.1. Copyright and License

6.2. Disclaimer

No liability for the contents of this document can be accepted. Use the concepts, examples and information at your own risk. There may be errors and inaccuracies, that could be damaging to your system. Proceed with caution, and although this is highly unlikely, the author(s) do not take any responsibility.