HelenOS installer
Having low-level fun with HelenOS

Dominik Táborský
Department of Distributed and Dependable Systems,
Charles University in Prague
dominik.taborsky@zoho.com

Introduction

HelenOS is a modern operating system built from scratch. One of its missing features had been the ability to self-replicate, i.e., to install itself on a hard disk from its own running instance. This work aimed at solving this problem by tackling all the individual subproblems. Please note IA32 and AMD64 are the only processor architectures considered.

For self-hosted installation few things are needed:

1. Partition the disk.
2. Create filesystem(s).
3. Install a bootloader.
4. Install the rest of the system.

Filesystems have already been covered and installing the system itself is, fortunately, simple - it only requires file copying. The rest is a problem, though. Partitioning and bootloader installation can only be performed from other operating system, e.g., a GNU/Linux distribution. Both can be worked around if HelenOS is not the only system to be installed on the host. But in the other case it is impossible to install HelenOS on its own. Therefore the main goal of this thesis is to provide tools sufficient for successful self-hosted HelenOS installation.

For partitioning Master Boot Record (MBR) and GUID Partition Table (GPT) labels were chosen. Master Boot Record is a legacy partition scheme designed in 1980s. It is probably the most widely used scheme to day. GUID Partition Table is the replacement for the old MBR. It has much cleaner design and offers more features, e.g., redundancy and checksums.

Offset	Length	Description
0	440	boot code
440	4	disk signature
444	2	zeroes (usually)
446	64	primary partition table
510	2	MBR signature

Table 1: The Master Boot Record sector description

Porting a bootloader was selected as preferred solution to developing a custom new one. Reasonable choice was GRUB Legacy as it is very widely spread, adheres to the Multiboot specification and has many useful features already. It mainly also supports many common operating systems.

Objectives

These are our main objectives:

1. Develop a partition editor.
2. Port a bootloader.
3. Set up an installation procedure of the rest of the system.

These are discussed individually below.

Methods

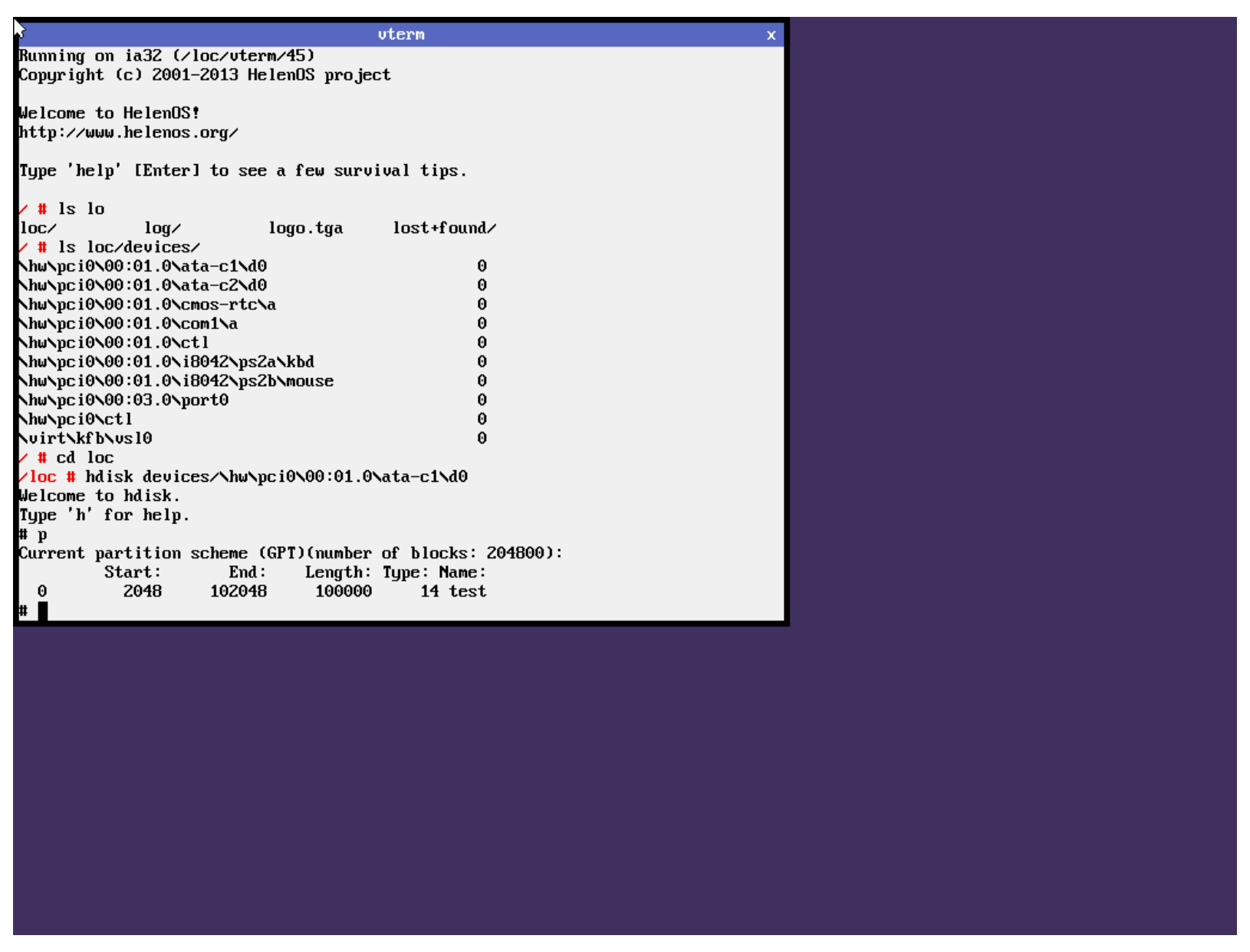
The implementation of the partition editor is to be split into parts: a single user-interface frontend and multiple backend libraries. This way we preserve the modularity of HelenOS, keep design clean and gain extensibility as a nice side-effect.

Offset	Length	Description
0	8	Signature
8	4	Revision number
12	4	Header size
16	4	Header CRC32
20	4	Reserved; must be zero
24	8	Current LBA
32	8	Backup LBA
40	8	First usable LBA
48	8	Last usable LBA
56	16	Disk GUID
72	8	Partition table LBA
80	4	Number of part. entries
84	4	Size of a partition entry
88	4	Partition table CRC32
92	*	Reserved; must be zeroes

Table 2: GPT header description

GRUB is used for bootloading HelenOS. Bootloader itself does not depend on any operating system. What does, though, is the installation of the bootloader. It has to be installed at the right place with the right values encoded. That tool exists only for POSIX-compliant systems, which HelenOS is not. Writing a new tool is possible, but requires deep understanding of the image-compilation process. Thus porting existing solution was chosen since in theory it only requires replacing library calls.

As of this time HelenOS uses GRUB to load and boot the kernel and load all required modules including the initial RAM disk (initrd image). This means there is no root filesystem on the hard drive. The RAM disk image is uncompressed into main memory and that area is used as the root. Therefore all system files cannot be simply copied over. Either the RAM disk is to be dumped into a file, thus recreating the module, copied over from the device the system booted off or all modules and kernel are to be put in the RAM disk image itself.



```
Running on ia32 (/loc/vterm/45)
Copyright (c) 2001-2013 HelenOS project

Welcome to HelenOS!
http://www.helenos.org/

Type 'help' [Enter] to see a few survival tips.

/ # ls lo
loc/          log/          logo.tga      lost+found/
/ # ls loc/devices/
\hw\pci0\00:01.0\ata-c1\d0                    0
\hw\pci0\00:01.0\ata-c2\d0                    0
\hw\pci0\00:01.0\cmos-rtc\a                   0
\hw\pci0\00:01.0\com1\a                       0
\hw\pci0\00:01.0\ctl                          0
\hw\pci0\00:01.0\i8042\ps2a\kbd               0
\hw\pci0\00:01.0\i8042\ps2b\mouse             0
\hw\pci0\00:03.0\port0                        0
\hw\pci0\ctl                                  0
\virt\kfb\vsl0                                0
/ # cd loc
/loc # hdisk devices/\hw\pci0\00:01.0\ata-c1\d0
Welcome to hdisk.
Type 'h' for help.
# p
Current partition scheme (GPT)(number of blocks: 204800):
         Start:      End:      Length:  Type: Name:
  0       2048      102048      100000      14 test
#
```

Figure 1: Running HelenOS and hdisk

Implementation

MBR and GPT were implemented as libraries *libmbr* and *libgpt*, respectively. They provide an interface that can be used in any other program. The partition editor is called *hdisk*.

Hdisk was designed to be only a user interface to both libmbr and libgpt. It is not, however, dependent on any of them. It is extensible by implementing a specific object and its methods, that is common to all label-manipulating libraries.

The user interface is simple, similar to that of *fdisk* known from GNU/Linux distributions. User inputs single letter to invoke each command which may request more input afterwards. Example of that is simply adding a partition. Some common commands are:

a Adds partition.
d Deletes partition.
h Prints list of commands.
p Prints current partition list.
r Re-reads partition records from the device.
q Quits hdisk.
w Writes the partition list to the device.

Hdisk is also extensible beyond the basic command set. Any label extension can implement command *e*, which may contain a new command set. Returning from this command resumes normal operation.

GRUB Legacy port required many small changes in many different places. These were mostly simple, though, requiring including different header files and slightly different function calls, e.g., replacing `strlen()` with `str_len()`. A major change was required to code simulating BIOS read/write functions. This had to be implemented using *libblock*, the *loc* service and IPC. Another change includes device name generation to allow looking up devices.

Finally, the easiest way to allow self-replication was chosen. The build procedure was adapted to output a RAM disk containing itself, the kernel and all the other modules. Although this change brought some disadvantages - doubled the RAM disk size and the loading time, it is simple and flexible enough for future improvements. When HelenOS no longer expects all root filesystem in RAM disk, or when device tree is stable enough so that all modules can be read from the install medium, then this adjustment is no longer needed and can be discarded.

Conclusion

All the objectives have been accomplished and the user now may install the system on a hard disk. Even though the solution is not optimal, HelenOS is expected to change substantially and therefore any complex solution might have come as a waste. However, hdisk and label-modifying libraries are well-written and useful in any case.

Possible improvements

Follow-up work may include porting GRUB 2, designing completely new partition scheme (similarly to, for example, BSD slices) or even implementing graphical partition editor.

HelenOS overview

HelenOS is an experimental, open source, general-purpose operating system built from scratch on top of its SPARTAN microkernel. It is a set of services, libraries and applications. One of its goals is portability so it runs on several different architectures. It also strives to be as modular as possible and eventually become a fully usable system.

Acknowledgements

I thank my family, my supervisor and anyone who supported me.