Linux headless

with PC Engines ALIX

Mathias Weidner

Linux headless

with PC Engines ALIX

Mathias Weidner

This book is for sale at http://leanpub.com/linux-headless

This version was published on 2014-02-22

Tweet This Book!

Please help Mathias Weidner by spreading the word about this book on Twitter!

The suggested hashtag for this book is #linux-headless.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#linux-headless

Contents

Preface

I have noticed that every personal computer costs - first and foremost - time. It doesn't matter whether it runs on MS Windows, OSX, Linux or UNIX, how much time it saves me or what - formerly unknown - possibilities it opens up. It takes time for the computer to boot up, time for me to log in, time to start a program. Or - even worse - time for finding something on the internet and copying it onto my computer. The PC can do nearly anything - allows me do nearly anything. But it also takes up my time and distracts me - because, while waiting for one thing, I can do this or that. Pretty soon I'm in multitasking mode and getting next to nothing done.

But this book is about something else. It's about small computers, built for one purpose: to be switched on and soon forgotten. It's about computers that don't have a keyboard or a display, that are there to do what they were built for. Sure such a device is boring when it works. But this book is not about using such devices, it is about building one.

- What do I need to do this?
- What do I have to consider?
- What can I do with such a device?
- What must I know in order to build it?
- What is already there?
- What kind of hardware can I use?

Certainly many things in this book will be outdated in short time, given the pace with which information technology evolves. Therefore I'll try to carve out some fundamental principals and take the available hardware as an example.

Audience

This book is for anyone who wants to build their individual computer according to their own design based on PC Engine ALIX or similar hardware.

It presumes a basic knowledge of the UNIX command line.

To build the UPS circuit, basic knowledge about electronics assembly is required.

Organization

The **first part** of the book deals with the things I can do or want to do with such a device. What software do I need and what special things do I need to consider? Problems will be dealt with in more detail in later chapters.

Chapter one illustrates some applications and talks about some basic conditions for these applications. These may include the number of interfaces for a networking device, additional hardware for special purposes or particular software requirements.

The **second part** deals with the hardware used. What does it have in common with personal computers, what is different?

Chapter two goes into detail about PC Engines ALIX computers. What hardware options are there? Which device drivers do I need? What software is available especially for this hardware?

Chapter three covers additional hardware for special projects.

The **third part** deals with the software used. It is about appropriate Linux distributions and how to install them onto the hardware. It also explains about the components of a Linux system and their purpose and about additional software and how to install it onto the system.

Chapter four goes into some Linux distributions that I find suitable for headless Linux projects.

Chapter five illustrates different ways of installing my choice of Linux distribution into the hardware.

Chapter six goes into detail about the components of a Linux system. What is their purpose and what alternatives do I have?

Chapter seven shows how I can compile and install software that is not available in my chosen Linux distribution.

The **fourth part** of the book deals with the daily operation and offers hints on how to solve possible problems.

Chapter eight goes into detail about daily operation. How do I configure the device for the operational environment? How do I update the system? How do I know if security updates are needed in the first place?

Chapter nine presents some trouble-shooting strategies. What can I do if a program doesn't start at all? When it crashes? How do I track problems in the network? Which programs help me with which problems?

Chapter ten introduces some protocols and mechanisms that I may encounter in the course of the project which may prove helpful to know about.

Chapter eleven lists some additional resources that may provide help for some problems.

Conventions

Program code and input in the command line is written in `constant width` text. This is also used inside paragraphs for literal command line options.

Emphasized text is written in *italics*.

Acknowledgements

This book began as accompanying documentation for a few projects with PC Engine ALIX computers.

I would like to thank the users of the PC Engines support forum for their helpful suggestions, especially Nicolas (*pure_debian*) for his reference to *flashybrid* and for the idea of using virtual machines.

Willy Tarreau suggested I use the PC8591 to monitor the mini UPS which made the circuit considerably easier.

Thanks go to KDG Wittenberg for providing me with some ALIX machines for testing purposes after my own device became operational and could only be used for a limited number of tests.

Many thanks to Lorri King from able Sprachschule for correcting my perception of how a meaningful english sentence should look. I take the credit for any remaining mistakes in English grammar and orthography.

Installing Linux

In this chapter I'll go into detail about different ways to install Linux for the first time on a machine.

Basically there are three ways:

1. I can prepare a CF boot medium on a second computer and insert this into the device.
2. I can boot and install from an USB stick. This only works for ALIX1.x as well as for ALIX.3D3 with AWARD BIOS. With the other models, in particular with TinyBIOS, I can't use USB as a boot medium.
3. I can boot and install using a network (PXE boot).

Of course it's also possible to combine these options.

Preparing a boot medium on a second computer

This approach requires a computer with the ability to read and write CF cards. I now have various options.

The easiest and fastest way is to take a prebuilt image, which I can transfer to the CF card with the *dd* command (or something similar on a MS Windows machine). This method allows me to clone a machine or - in case of hardware failure - restore a machine really quickly. I might have to adapt some udev rules with regard to network adapters because these are bound to the hardware address on some distributions.

As an alternative to cloning an image with dd I can create the partitions and file systems manually, mount the file systems on the second computer and install the operating system using *debootstrap*, *tar*, *cpio* or something similar. Afterwards I have to install the boot loader on the CF card using *grub-install*.

Another possibility is to prepare the prospective system as a virtual machine (VM) and test everything including the serial console. When the system is ready I copy the VM “raw”, that means bit for bit, onto the CF card. If I choose this option, I have to consider a few things:

- When creating the VM, it should be fully virtualized. Paravirtualization requires an environment that is not available on an ALIX board. In the best case I’ll have installed too much software, in the worst case the system won’t work at all.
- The hard disk of the VM shouldn’t be larger than the CF card.
- Some hardware assignments, for instance the binding of an ethernet interface to a specific MAC address, may have to be opened in the *udev* rules.

I myself have not prepared a system in this way, but I consider it to be a viable approach if you want to alter a running system and there is no hardware available to test the changes thoroughly before deployment.

At the end, the CF card, which has been prepared in one of the different ways, is inserted into the ALIX machine and the device is ready to use.

Preparing OpenWrt on another computer

OpenWrt allows me to easily prepare a CF card on another computer. There are different images on the download pages which I can write onto the CF card using *dd* and then insert into the machine. The *x86-generic* images with an ext2 file system are suitable for ALIX machines, for instance *openwrt-x86-generic-combined-ext2.img.gz*. This image contains a master boot record, a small partition with the boot loader *grub* and a partition with the root file system. After I have written the image onto the CF card, I can increase the size of the root partition using *gparted* or *parted*. Alternatively I can create another partition with another system on the CF card for experiments and add appropriate entries in the grub configuration of the first partition.

When preparing the CF card on another computer I have to open the case of the ALIX machine and then insert the card afterwards. Using the approaches below I can insert the CF card into the machine beforehand and directly install the system, using the ALIX machine itself.

Installation using USB media

In the iMedia Linux forums I found the following procedure for installing the system from a USB CD-ROM onto an ALIX board with TinyBIOS:

1. Download a minimal CF archive from http://resources/imedialinux.com called *imedia-cfboot-x.y.zip*, where x.y stands for the version number.
2. Format a CF card with an ext2 file system.
3. Unpack the CF archive into the file system on the CF card.
4. Install the boot loader *GRUB* on the CF card.
5. Burn the installation CD-ROM and connect a USB CD-ROM drive to the ALIX board.
6. Insert the CF card into the ALIX board and start it. The mini-operating system on the CF cards starts, mounts the CD-ROM and installs from the CD-ROM.

On ALIX.1 you an alternatively try to create a USB live system with *UNetbootin* and start the machine with it. I haven't yet tested this sort of ALIX.1 board.

Installation using PXE-Boot

In my opinion the most elegant and flexible solution is to install the system with PXE boot via a network.

Here the computer loads the boot loader from the network. The boot loader loads the programs needed for the installation and these install the operating system. This way I can work with a completely assembled machine and don't have to open the case before or after installation. I only have to take care that the machine starts in PXE-boot mode.

Tip

On a PC Engines ALIX I have to press `n` on the serial console while it is testing the memory in order to boot in PXE-boot mode once.

If I press `s` instead, I enter setup mode. There I can press `e` to set PXE as permanent boot mode. Afterwards I exit setup by pressing `q` and confirm my changes. I can unset the PXE boot the same way.

One PXE-boot installation is not like the other. There are various ways to individualize them or automate them.

Standard Debian installation using PXE-boot

Here the boot loader starts a mini system containing the Debian installer.

I first prepare a DHCP server for a PXE-boot by using ISC DHCPd. I can do this for one host depending on the number of devices, I want to install:

```
host hostname {
  next-server bootservername;
  filename "bootladerfile";
  hardware ethernet 01:02:03:04:05:06;
}
```

Or I can do it for a group of similar devices:

```
group {
  next-server bootservername;
  filename "bootladerdatei";
  host hostname1 {
      hardware ethernet 01:02:03:04:05:06;
  }
  host hostname2 {
      hardware ethernet 01:02:03:04:05:07;
  }
}
```

The entry *bootservername* refers to the TFTP server for the boot loader. This TFTP server should understand the option *tsize*, for instance *tftp-hpa*. I use **pxelinux.0** from *PXELINUX* as a *bootloader file* which is part of the *SYSLINUX* project. I put the files of the Debian installer in a directory tree under the TFTP root file system. The configuration for *PXELINUX* is in a file in the directory *pxelinux.cfg* under the TFTP root and looks like this for the Debian installer:

```
SERIAL 0 38400 0
DEFAULT install
LABEL install
  kernel d-i/i386/linux
  append initrd=d-i/i386/initrd.gz -- \
         console=ttyS0,38400n80
```

In this example I stored the files of the Debian installer in the directory *d-i/i386*. (The line break before `console=` is just for the formatting of this book. This line belongs to the `append` line.)

This installation is interactive just like a normal installation from a CD-ROM. I have the same options within the constraints of the hardware. Because the installation is interactive, it also ties up my time. There is little else I can do in the meantime except use *Preseed* to respond to the questions of the Debian installer in the debconf database.

Installation using PXE-Initrd

In my opinion, using *pxe-initrd* is a little better. This is a collection of scripts and configuration files which allow you to boot a Linux system using PXE, format the hard disks and copy the operating system afterwards from a server using *rsync*. Everything works fully automatically without the need for interaction.

Tip

I have different files for *localboot* and *pxe-initrd* on the TFTP server in the directory *pxelinux.cf*.

The file named *default*, which is used for all systems that don't otherwise fit has this content:

```
default localboot

  label localboot
  localboot 0
```

This configuration means the machine boots the locally installed system, in case the computer accidentally boots with PXE enabled.

For all systems where *Voyage Linux 0.7.5* is to be installed the file *voyage-0.7.5.conf* contains the following:

```
serial 0 38400
console 0
label linux
    KERNEL vmlinuz-2.6.38-voyage
    APPEND initrd=initrd-2.6.38-voyage \
          console=ttyS0,38400n1 \
           root=/dev/hda1 root_size=768
```

It uses this to load the kernel and the initramfs for the *Voyage Linux* version. (The line break before `console=` and `root=` is just for the formatting of this book. Both lines belong to the `APPEND` line.).

In a posting from 2009 on the mailing list *[voyage-linux]* I found a note on *jra-initrd* from Jeff R. Allen. Back then he had taken *bit-pxe* and modified it to fit his needs. When I tested his scripts with Voyage Linux version 0.7.5 I found that his scripts no longer worked. I adapted the scripts from *jra-initrd* in October 2011, customized them to fit my needs and published them as *pxe-initrd* (please see the appendix for sources). These scripts do work with version 0.8, but they still have to be tested with newer versions.

The operating system you want to install is stored as a directory tree on a server and made available through an *rsync* server. The scripts on the initramfs format the local CF card, mount the file systems and copy the operating system using *rsync*. Afterwards the boot loader *grub* is installed and the machine reboots. No interaction is necessary to do all this and everything is determined beforehand. All I have to do is to make sure that the operating system doesn't get installed again after rebooting. Either I only start the ALIX machine once with a PXE boot by pressing `n` during the memory test or I change the PXELINUX configuration on the TFTP server to `localboot` after installation has begun.

Pxe-initrd has the advantage of being quick to setup and I don't need a complex infrastructure. Instead, in the simplest case I need *pxe-initrd*, a directory tree with the operating system I want to install, *dnsmasq* and *rsync* as well as a free ethernet interface if I don't want to install the new machines in the production network. The installation itself needs absolutely no interaction. Once it is started I'm able to do something else and come back later. To a certain extent I can control the installation retrospectively, that means at boot time with kernel parameters. And I can modify the directory tree containing the operating system by doing a `chroot` in the root directory and modifying it from within.

This has some disadvantages. For instance the modification is a little bit cumbersome since I have to do a `chroot` in the root directory, where I do not have the whole environment like a normal system (*procfs*, *sysfs* and device files are lacking in the chroot environment). For most changes I do not need these, for others there are workarounds (such as additionally mounting these file systems in the chroot environment). Some software packages can only be installed using a few tricks. Furthermore I need complete directory trees for all of the systems I want to install and therefore more hard disk space.

Tip

The *postinit* script of the *busybox-syslogd* package tries to start the demon program which fails in the chroot environment. Thus installation is incomplete, something which I can't repair easily with `apt-get -f install`. In this case the following trick will help:

```
# chmod -x /etc/init.d/busybox-syslogd
# chmod -x /etc/init.d/busybox-klogd
# apt-get -f install
# chmod +x /etc/init.d/busybox-syslogd
# chmod +x /etc/init.d/busybox-klogd
```

If I modify both scripts to prevent the demon programs from starting, the *postinit* script doesn't start these programs which ensures an error-free installation.

PXE installation with FAI

FAI (Fully Automated Installation) is the fine art of automatic installation of Debian based Linux. This installation system is well-engineered and therefore needs quite a bit of time to comprehend, especially if you want to exploit all its possibilities. But it is worth the time if you have several different systems and regularly want or have to setup new systems.

Together with other tools for automating system administration tasks, like *cfengine*, *puppet* or *chef* it makes the daily routine of the system administrator tremendously easier, but I digress.

FAI's main advantage over pxe-initrd is its extreme flexibility. Completely interaction free installation is possible because the installed systems at the end of a successful installation automatically change their configuration at the TFTP server to *localboot*.

The systems you want to install aren't adapted individually. Instead you define classes that match the systems requirements and the system gets installed according to the defined classes. Everything can be planned beforehand and FAI installs the system according to the plan.

There are a few disadvantages, which your project should take into consideration. You will need the right infrastructure, including a DHCP server, a TFTP server and an additional NFS server. These three are often combined in the so-called FAI server. The time you'll need to become familiar with FAI adds to the project time. Everything has to be planned beforehand. FAI only installs the plan.

All in all, in my opinion FAI is nevertheless the silver bullet. If you are already using it or intend to use it anyway for other machines, just give it a try.

In the end, everyone has to decide for themselves how to get their Linux system onto the ALIX machine in order to move on to the next steps of their project.

Colophon

This book was written in the *Markdown* format with some extensions from leanpub. Then Leanpub[1] used the Markdown sources to create the PDF, EPUB and MOBI versions.

The circuit for the UPS in chapter 3 was created using *gschem* from the gEDA project and then converted to PNG.

All images were processed using the GNU Image Manipulation Program (GIMP).

The device on the cover page is an open-case ALIX 2D3 with a CompactFlash card inserted in the card drive. I used these devices to test what I wrote in this book.

[1]https://leanpub.com/