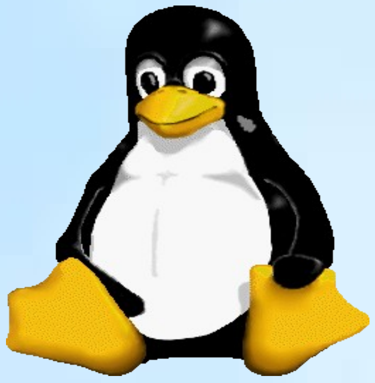
Create your own Linux

Limux the Linurs Mini Linux



1st preliminary edition still in preparation

2014-01-01 by urs@linurs.org

Latest published version of this document:
http://www.linurs.org

Limux (Linurs Mini Linux)

Contents

An other Linux distribution?

The goal is not to create an other Linux distribution. The goal of this document is to have a minimalistic Linux that boots fast and can be used on small embedded PC's. It is therefore a good point to start an embedded Linux project, where additional applications can be added to create a Linux device. Last but not least, starting from scratch, is a good way to learn how Linux works!

The Linurs Mini Linux (Limux) has been created on a computer running Gentoo Linux http://www.gentoo.org, however it should be easy to do the same on a computer running an other Linux distribution.

The goal is to start simple, then expand and customize your Linurs Mini Linux step by step to have your own Mini Linux that serves your needs.

The document assumes that you have some Linux and some Gentoo Linux knowledges and since I do not want to repeat myself you should know my Linux book see: http://www.linurs.org/linux.html

This Limux (Linurs Mini Linus) has nothing to do with: LiMux Die IT-Evolution
I'm a hobby programmer and LiMux is a project name of the city München to move to Linux.

1. General items

Important: Whenever possible, try to not work as root to minimize the risk of damaging your system. You want to create a new system and not to modify your existing system!

A complicated issue is when the CPU between system to be used to develop and system of the embedded device are not binary compatible. All binaries have to be cross compiled. Therefore it is a good strategy to do the first tests on a binary compatible CPU and even better use the PC, where you probably read this document.

A Mini Linux requires as any other Linux system a root file system, a kernel and something to boot (the boot loader). The root filesystem contains all files and directories that the kernel can access. The kernel and bootloader can be also inside the same filesystem, but they could be somewhere else.

2. busybox

The application for small Linux systems is busybox. Instead of having all individual programs to allow you working in a console, busybox can be configured to have just on single binary file for that.

Busybox should is already on the Gentoo system but the sources are missing.

To customize busybox the sources must be used. To not get in conflict with your Gentoo installation, just download and unzip them from:
http://busybox.net/downloads

As regular user (to avoid conflicts with your system), open a console and cd to the downloaded and unzipped busybox directory.

Instead of starting from zero type `make defconfig` to get a default configuration. After that type `make menuconfig` to get a menu similar to the one of the kernel configuration.

Under the Busybox Settings > Build Options select with Build BusyBox as a static binary (no shared libs) to get everything in one file. Then do `make` to get the result.

Installing busybox creates a root filesystem, so take care to not overwrite the root filesystem of your host! To avoid mistakes don't do the following as root, do it as regular user. It installs busybox into a directory where the root system of the Mini Linux will be created:

```
make CONFIG_PREFIX=<path to a directory> install
```

Take a look at the directory to see what has been created. Basically lots of links with the familiar names that point to /bin/busybox that is the only executable. The single executable finds easily out from what command line it has been called. Since the calling link has the name of what the user wants to do, busybox knows what it has to do. Type `ldd busybox` to verify that busybox does not depends on libraries, but is a single binary.

To understand all busybox commands and what is different to full blown systems see http://busybox.net/downloads/BusyBox.html

3. Systems without busybox

It is not a must using busybox. The Mini Linux could start a script or any executable. However then you have to make sure that all files require and depend on the script exist. As example if you like to run a script, it can not run without bash, so copy bash to the root file system.

```
whereis bash
bash: /bin/bash /etc/bash /usr/man/man1/bash.1.bz2 /usr/share/man/man1/bash.1.bz2
```

Note: bash can not run, since its libraries are missing. Type

```
ldd /bin/bash
linux-gate.so.1 =>  (0xffffe000)
libncurses.so.5 => /lib/libncurses.so.5 (0xb7edc000)
libdl.so.2 => /lib/libdl.so.2 (0xb7ed8000)
libc.so.6 => /lib/libc.so.6 (0xb7d95000)
/lib/ld-linux.so.2 (0xb7f47000)
```

and you'll see what you have to copy to the /lib directory of the root file system. For something well known that is very simple and has to be very small, it might the way to go, but using busybox is less work and more flexible. If you want something complex, then take a real distribution as http://www.gentoo.org. The book assumes therefore that busybox will be used for the Mini Linux.

4. The initial RAM disk

See for system without initrd and so on
http://www.gentoo.org/doc/en/liveusb.xml
A Mini Linux requires as any other Linux system a root file system. One of the simples way is not having the file system on a physical device itself and use device drivers to access it, but use the RAM itself for that. Such root file system for the RAM are called initrd (Initial RAM Disk).

A maybe drawback is that the when editing a file later its changes get lost after next boot. The edited file is in RAM and is not written back to the initial RAM disk. For embedded systems this drawback might be an advantage, since a reset starts with the clean and tested initial RAM and makes your system reliable and robust.

4.1. Create the initial ram disk

Open a console `cd /boot` it is where the initial ram disk has to go. In an other place as ~/mylinux your minimalistic linux system will be prepared:
Type `su` to become root privileges:

Create a ext2 initial ram disk with a given size, smaller that the CONFIG_BLK_DEV_RAM_SIZE=4096 defined in the kernel, but make sure the size is bigger than the stuff that is copied there later on!

```
dd if=/dev/zero of=myinitrd bs=3000k count=1
```

Now format it as ext2

```
mke2fs -F -m0 myinitrd
```

Create the mounting point `mkdir /mnt/mylinux` and now mount it

```
mount -t ext2 -o loop myinitrd /mnt/mylinux
```

And copy the root system, that for instance busy box has created over

```
cp -R <directory>/* /mnt/mylinux
```

Busybox did not create everything needed for the root file system.

The following steps are the key steps how your Linux system behaves. Since frequently modifications, improvements and adaption are expected, the text might be outdated and you should better look at the files in Limux available via http://www.linurs.org/limux. However to still have some text here, to show what has to be done, to learn and to see the complexity, I decided to have some sample code here, even knowing that it is probably outdated:

????????? Update the following, once I'm happy, provide a script and sample files

Create some directories (alternatively you could to it in the directories that busybox has created and then copy it over) :

```
mkdir /mnt/mylinux/dev
mkdir /mnt/mylinux/proc
mkdir /mnt/mylinux/sys
mkdir /mnt/mylinux/etc
mkdir /mnt/mylinux/mnt
mkdir /mnt/mylinux/tmp
mkdir /mnt/mylinux/var
mkdir /mnt/mylinux/var/log
mkdir /mnt/mylinux/var/run
mkdir /mnt/mylinux/lib
mkdir /mnt/mylinux/lib/modules
```

Some device files are required, copy device files might be problematic, so it is easy to create them here:

The console device:

```
mknod /mnt/mylinux/dev/console c 5 1
```

And the ttys to make the default busybox happy:

```
mknod /mnt/mylinux/dev/tty1 c 4 1
mknod /mnt/mylinux/dev/tty2 c 4 2
mknod /mnt/mylinux/dev/tty3 c 4 3
mknod /mnt/mylinux/dev/tty4 c 4 4
```

The block device file where the boot partition is for a IDE hard disk it is hda with the major number 3 and its first partition 1

```
mknod /mnt/mylinux/dev/hda1 b 3 1
```

adapt it when you have other devices as serial ATA device

```
mknod /mnt/mylinux/dev/sda1 b 8 1
```

Create the /mnt/mylinux/etc/fstab file

```
none        /dev/pts    devpts      defaults    0       0
none        /proc       proc        defaults    0       0
none        /sys        sysfs       noauto      0       0
```

The /mnt/mylinux/etc/group file

```
root:x:0:
users:x:100:
```

And the /mnt/mylinux/etc/passwd file

```
root:*:0:0:root:/:/bin/sh
```

A small /mnt/mylinux/etc/inittab

```
tty1::askfirst:-/bin/sh
tty2::askfirst:-/bin/sh
tty3::askfirst:-/bin/sh
::ctrlaltdel:/sbin/reboot > /dev/null 2>&1
::restart:/sbin/init
```

Create a link to have an /mnt/mylinux/etc/mtab

```
ln -s /proc/mounts /mnt/mylinux/etc/mtab
```

And finally the init script that will be started

```
#!/bin/busybox sh
mount /proc
mount /sys
mount /dev/pts
# Load all kernel modules
VER=$(uname -r)
for m in $(cat /lib/modules/$VER/modules.*map|cut -d" " -f1|sort -u); do
  modprobe $m
done
export TERM_TYPE=pts
exec /bin/busybox init
```

Note: all those files above need to end with a <CR> character, so you must be able inside the editor to place the cursor on a line below, otherwise the last line might not be executed.

Now everything should be there, so unmount it

```
umount /mnt/mylinux
```

and zip it

```
gzip -9 myinitrd
```

????????? To be tested alternative way to create the initial ram disk
find . | cpio -o -H newc | gzip > /boot/mylinux.img

The file will use the (SVR4) portable format, which supports file systems having more than 65536 i-nodes. Can I moun't such an image???

4.2.Edit and modify the initial RAM disk

To see what is inside you can always open a console as root and unzip it

```
gunzip myinitrd.gz
```

then

```
mount -t ext2 -o loop myinitrd /mnt/mylinux
```

Edit it and when done

```
umount /mnt/mylinux
```

and do not forget to zip it again

```
gzip -9 myinitrd
```

5. The Kernel

To keep it simple, the kernel should have all modules integrated that are needed during boot. Otherwise modules in /lib/modules must be loaded using a script.

The location of the kernel must be known, following the gentoo installation handbook it is on the first hard disk on the first partition that is mounted to /boot (The boot directory of the 3rd partition of the same device, is the mounting point of the first partition). Since boot loaders might be restrictive on filenames, avoid problems and use a short name for the kernel following the old DOS rules: 8 characters.

The initial ram disk must be at the same harddisk and partition as the kernel, so put it there as well. When you followed the previous instruction the /boot/myinitrd.gz file is already there if not copy it.

The kernel to be used must be able to mount the initial ram disk. Therefore check if the kernel supports the following:
CONFIG_BLK_DEV_RAM and CONFIG_BLK_DEV_INITRD.

```
cat /usr/src/linux/.config | grep CONFIG_BLK_DEV_RAM
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_RAM_COUNT=16
CONFIG_BLK_DEV_RAM_SIZE=4096
cat /usr/src/linux/.config | grep CONFIG_BLK_DEV_INITRD
CONFIG_BLK_DEV_INITRD=y
```

If you do not get such responses create a new kernel with: Device drivers > Block devices CONFIG_BLK_DEV_RAM and General setup CONFIG_BLK_DEV_INITRD.

6. Running your Linux from a chrooted environment

Instead of rebooting, burning a CD, preparing a USB Stick or starting a virtual machine, you can test your initial ram in a console on your host. No bootloader is required (or tested), the kernel on the host is used.

The initial ram disk needs to be mounted, so

```
cd /boot
```
and
```
gunzip myinitrd.gz
```
then
```
mount -t ext2 -o loop myinitrd /mnt/mylinux
```

Do not loose the devices and proc filesystem
```
mount -t proc none /mnt/mylinux/proc
mount -o bind /dev /mnt/mylinux/dev
```
And now change the root using later /bin/sh as the shell
```
chroot /mnt/mylinux /bin/sh
```

When done
```
exit
```
and do not forget
```
umount /mnt/mylinux/proc
umount /mnt/mylinux/dev
umount /mnt/mylinux
```
and do not forget to zip it again
```
gzip -9 myinitrd
```

7. Running your Linux from a hard disk

Now the Mini Linux can be started on your host PC that runs already Linux or any other hard disk. In this first attempt, it uses a kernel and the boot loader that are already on your PC.

7.1. Prepare the bootloader

Modify /boot/grub/grub.conf

A grub entry that starts the shell after the kernel is up:
```
title=Mini Linux 2.6.30-gentoo-r4-2009-08-06 bb shell
root (hd0,0)
kernel /kernel-2.6.30-gentoo-r4-2009-08-06 root=/dev/ram0 init=/bin/sh rw
initrd /myinitrd.gz
```

A grub entry that starts the busy box after the kernel is up:
```
title=Gentoo Linux 2.6.30-gentoo-r4-2009-08-06
root (hd0,0)
kernel /kernel-2.6.30-gentoo-r4-2009-08-06 root=/dev/ram0 init=/linuxrc rw
initrd /myinitrd.gz
```

8. Running your Linux from a memory device

After power up, the motherboards Bios comes alive and tries to load and start a program that lies on a specific location on the boot device.
This specific location on the selected memory device is the boot sector. The reason behind this is that the Bios does not understand a lot about file systems. Copying data

to a device should never touches this boot sector. A special procedure or program has to be used for that. Fortunately grub offers all that for you.

The memory device can be a USB stick, CompactFlash, SD card or any other device. The following assumes that it is a USB memory stick.

First make sure your motherboard allows to boot from the usb device. Plug in your USB stick and reboot your PC, go into the BIOS and check the harddisks in the boot sequence menu (or the removable media). If your USB stick appears there then it is ok, otherwise give up and skip this chapter or look for an other computer.

8.1.The Windows warning and the USB stick partitions

You probably want to start and create a Linux partition on your memory stick and copy all your stuff onto it. But be warned, if you or your friend, plugs this memory stick into a windows (e.g. XP or l'abbiamo vista) computer, the windows computer probably comes with the following question: Not formatted USB disk. Do you want to format? If you or your friend answers yes, then all your work is gone and you can say thank you Bill you are my friend!

To avoid calling Bill your friend, I suggest to use a memory stick with a FAT32 partition. It is horrible to work under Linux on a FAT32 partition, since it is not compatible with the owner and group permission Linux uses and the wonderful feature of Linux to use symbolic links and other stuff. However we work with the initial RAM disk, so the computer will see the FAT32 partition just during boot, after that we will have a ext2 partition holding our root file system.

An other way would be to create two partitions on your memory stick. The first is bootable and holds a FAT32 partition (be aware about some restrictions as Linux links). On this first partition you can put grub and maybe the linux kernel and the initial ramdisk file initrd. The second partition will be ext2.Windows will not see this partitions, however it shows the whole memory stick size.

It is recommended to check how the USB stick is formatted and partitioned:
Check that it is not mounted and do something as `fdisk /dev/sdf`
Verify that it is a single clean FAT32 partition and the partition is bootable (* character). W95 FAT32 (LBA) with its id c is a good choice.
And if not how to create and format a USB stick with fat32?

For just something that always runs on Linux and never gets in touch with the Microsoft world, use ext2. After having run `fdisk /dev/sdf` and made a bootable ext2 partition, do `mke2fs /dev/sdf1` then mount it (plug in and out when run hal). To see where it is mounted `cat /etc/mtab`

8.2.Installing grub

Create on the first (FAT) partition a /boot/grub directory. To install grub copy from your host PC that runs grub the files stage1, stage2 and grub.conf (and maybe the splash image splash.xpm.gz) to your memory sticks /boot/grub directory. Since on your host PC menu.lst is a link to grub.conf and links do not work on a FAT32 partition, rename grub.conf to menu.lst on your USB stick. Edit then the file menu.lst to your needs:

```
default 0
timeout 5

title=Linurs Mini Linux
root (hd0,0)
kernel /kernel root=/dev/ram0 init=/linuxrc rw
initrd /myinitrd.gz
```

Now everything is there however, the memory device needs a boot sector:

I assume that you run now from a Linux computer where its internal disk and first partition (hda1) is called by grub hd0,0. Your usb memory sticks first partition is hd1,0.
To make grub on the USB stick bootable start on your Linux computer `grub`.

Note: Use the TAB key completion feature of grub, as example type `root (hd` TAB to see what you can select.

When the prompt comes type in the following:

```
root (hd1,0)
setup(hd1)
quit
```

8.3.Booting the memory stick

During next boot go into he BIOS and modify the boot sequence of your hard drive.

9. Running your Linux from a CD

9.1.Create a directory iso

Change to the ISO directory and copy the kernel and the initial ram disk there. Note iso9660 is restrictive on filenames so use short file names.

9.2.Getting isolinux

Isolinux is a boot loader for CD's. It is delivered with the package syslinux that contains other boot loaders as well.

9.2.1.From a live cd

Most Linux live CD's as Knoppix http://www.knoppix.org has it.

Limux (Linurs Mini Linux)

Therefore place a live CD in your cd drive.

9.2.2.From the original site

that has also a manual
http://syslinux.zytor.com/wiki/index.php/ISOLINUX

9.2.3.The gentoo way

```
emerge syslinux
```

then you find it in /usr/share/syslinux/isolinux.bin

9.3.Install isolinux

Copy the stuff needed: isolinux.bin and maybe also isolinux.cfg into the iso directory. Make sure isolinux.bin is executable. Now edit (or create) isolinux.cfg to have the following:

```
# My Linux boot-CD
# Show content of message.txt
display message.txt
# Show bootpromt
prompt 1
# Wait for 10 seconds
timeout 100
# If nothing entred go to the default label
default mylinux

  label mylinux
  kernel kernel
  append ramdisk_size=100000 init=/linuxrc initrd=myinitrd.gz
```

Create the message.txt file and put something in as:

```
****************
       Limux
         the
Linurs Mini Linux
  www.linurs.org
****************
```

9.4.Create the ISO image

Now the magic command to convert the files to an ISO image (using cdrtools)

```
mkisofs -o /home/lindegur/mylinux.iso -b isolinux.bin -c
boot.cat -no-emul-boot -boot-load-size 4 -boot-info-table
-r -V mylinux /home/lindegur/iso/
```

To check it you can:

```
mount -o loop mylinux.iso /mnt/mylinux
```

Now it is time to think about the environment and take a CD-RW and burn the ISO image using your favorite CD burning application as 3kb.

Some isolinux links:
http://syslinux.zytor.com/wiki/index.php/ISOLINUX

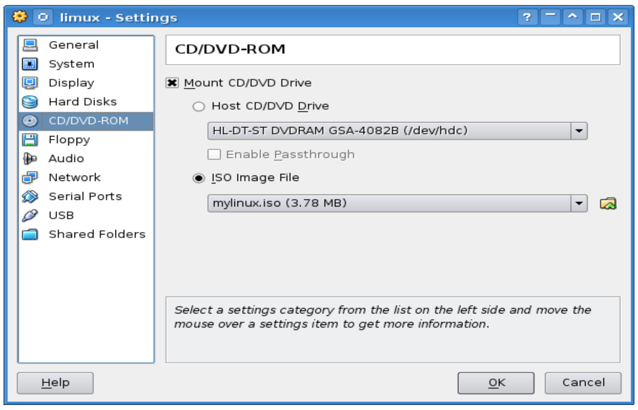
Limux (Linurs Mini Linux)

http://www.kernel.org/pub/linux/utils/boot/syslinux/
http://www.linuxroute.de/howto/isolinux_bootcd.php

the "-l" or "-iso-level 2" option to mkisofs to generate long (up to 31 characters) plain filenames.

10.Running your Linux inside a virtual machine

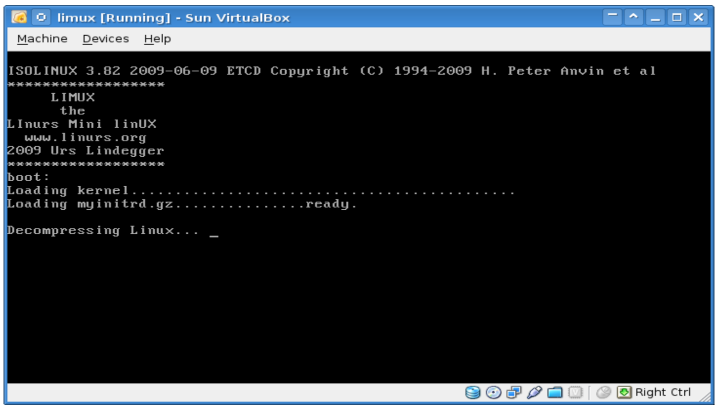
When testing Limux, it is time consuming to reboot the PC every time a new version is created or burn CD-RW's. It is more efficient to use a virtual machine as provided by Virtual Box for that. Type `VirtualBox` to start it. In Virtual Box create a Linux virtual machine and configure it via its GUI and the wizard. Under the CD/DVD-ROM menu, add the ISO image and start the virtual machine.


limux - Settings

CD/DVD-ROM

[✓] Mount CD/DVD Drive
○ Host CD/DVD Drive: HL-DT-ST DVDRAM GSA-4082B (/dev/hdc)
☐ Enable Passthrough
● ISO Image File: mylinux.iso (3.78 MB)

Select a settings category from the list on the left side and move the mouse over a settings item to get more information.


limux [Running] - Sun VirtualBox

```
ISOLINUX 3.82 2009-06-09 ETCD Copyright (C) 1994-2009 H. Peter Anvin et al
******************
     LIMUX
      the
LInurs Mini linUX
  www.linurs.org
2009 Urs Lindegger
******************
boot:
Loading kernel.........................................
Loading myinitrd.gz...............ready.

Decompressing Linux... _
```

If you click into the window where Linmux is running VirtualBox probably captures your mouse and keyboard. Per default press the right `Ctrl` key to let the mouse leave the window.

11.Running from a floppy emulation or a real floppy

With the floppy emulation you need just a Limux image file and a grub entry to be maintained. This makes floppy images still attractive even though floppies are outdated to day.

Use a real floppy and do a low level format:

```
fdformat /dev/fd0
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... done
emerge sys-fs/dosfstools
```

Edit /etc/mtools/mtools.conf

```
mkdosfs /dev/fd0
syslinux /dev/fd0
mount /dev/fd0 /mnt/floppy
```

Create the syslinux.cfg file:

```
DEFAULT linux
LABEL linux
SAY Now booting the kernel from SYSLINUX...
KERNEL kernel
APPEND ramdisk_size=100000 init=/linuxrc initrd=myinitrd.gz
```

Add now the kernel and myinitrd.gz to the floppy. Note everything needs to be less the 1.44M Byte, so if you have a fat kernel, there is no way to go.

??????????the kernel is too big

??????????? how to create a floppy image, (to consider linux knows also floppies so a linux and not a microsoft floppy using grub might be used as well)? Memdisk supports also harddisks

Copy the floppy image to /boot and add in your grub.conf:

```
title=Limux Floppy Image
root (hd0,0)
kernel /memdisk
initrd /myfloppy
```

memdisk is the floppy emulator and can be obtained by `emerge syslinux` and `cp /usr/share/syslinux/memdisk /boot`. Nice but how to add stuff?

Create a mounting point `mkdir /mnt/floppy` and mount the image:

```
mount -o loop /boot/myfloppy /mnt/floppy
```

12.Customizing Limux

13.Cross compiling

14.Annex A: Bibliography

Books in German (Writing this chapter, I found out, that I do not have English Linux books that I can recommend):

[1] Linux
Installation, Konfiguration, Anwendung
Michael Kofler ISBN 3-8273-1854-8

[2] Gentoo Linux
Die Metadistribution
Tobias Scherbaum ISBN 978-3-8266-1769-0

[3] C und Linux
Die Möglichkeiten des Betriebsystems mit eigenen Programmen nutzen
Martin Gräfe ISBN 3-446-22055-0

[4] Steuerungsaufgaben mit LINUX lösen; Eine Einführung anhand praktischer Beispiele
Andreas Zickner ISBN 3-7723-5109-3

[5] Messen, Steuern, Regeln mit Linux
Einsatzmöglichkeiten für Linux in Embedded Systems
Klaus-Dieter Walter ISBN 3-7723-4484-4

[6] Embedded Systeme mit Linux programmieren
GNU-Softwaretools zur Programmierung ARM-basierender Systeme
Edmund Jordan ISBN 3-7723-5599-4

[7] Linux
Das distributionsunabhängige Handbuch
Johannes Plötner Steffen Wendzel ISBN 3-89842-677-7

[8] Gentoo Linux
Installation – Konfiguration – Administration
Gunnar Wrobel ISBN 978-3-937514-34-5

[9] c't magazine 2009 issue 12,
This article and a project in work, inspired me to create this document.
Mirko Dölle http://www.ctmagazin.de/0912156

Alphabetical Index