A Bootable Embedded-Linux USB Flash Drive – HOWTO

By: Wiwat Tara, Email:ohhohe@gmail.com March 22, 2005

Requirement

- USB Flash Drive
- Host: x86 Fedora Core 2 or other Linux distros
- Target: x86 with bootable USB port
- Bootloader: GRUB
- Root filesystem: cramfs, ext2
- Linux Kernel Source: 2.6.10
- Busybox 1.0
- Coyote 2.21

Building

- Bootloader: Installing GRUB into MBR of USB Flash Drive
- Linux Kernel: Compile Linux Kernel 2.6.10
- Root File System: make cramfs (unwritable) root filesystem and ext2 for writable storage
- Application: telnetd, udhcpc, httpd, sshd, busybox

Overview of Flash Drive Partitions

1	2	3	
Boot Loader (ext2)	root file system (cramfs)	writable (ext2)	
5 MB	8 MB	50 MB	

Host

On the host machine's home directory, make a directory name ~/usb_linux/ to contain all necessities. Then create these directories and files:

```
$ ls -l ~/usb_linux/
total 1496
-rwxr--r--   1 root root     112 Mar 16 13:42 backup_usb_writable_partition.sh
drwxr-xr-x   2 root root    4096 Feb 22 11:15 bootldr
-rw-r--r--   1 root root 1474560 Mar  1 16:28 bootusb-0.8.img
drwxr-xr-x   4 root root    4096 Mar 15 11:26 build-tools
drwxr-xr-x   2 root root    4096 Mar  2 10:57 document
drwxr-xr-x   2 root root    4096 Mar  9 10:43 images
drwxr-xr-x   4 root root    4096 Mar 10 09:32 kernel
drwxr-xr-x   2 root root    4096 Feb 22 11:16 project
drwxr-xr-x  11 root root    4096 Mar 22 13:14 rootfs
drwxr-xr-x   4 root root    4096 Mar 15 09:18 sysapps
drwxr-xr-x   3 root root    4096 Mar  1 08:56 tmp
-rwxr--r--   1 root root      92 Mar  9 17:01 update_flash_cramfs.sh
drwxr-xr-x   6 root root    4096 Mar 16 13:44 usb
```

And on my x86 host machine it detected a USB Flash drive as a scsi device (see below.) The first scsi device here is the harddisk drive (/dev/hda) and the second one is actually the USB flash drive (/dev/hdb) This is true until you attempt to boot off the USB Flash drive because as soon as the system is up, the very first scsi device detected by kernel is USB Flash and it will be given /dev/sda instead. But now we are on the host doing development.

```
$ cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: ATA      Model: Maxtor 6Y120M0   Rev: YAR5
  Type:   Direct-Access                    ANSI SCSI revision: 05
Host: scsi2 Channel: 00 Id: 00 Lun: 00
  Vendor: IBM      Model: Memory Key       Rev: 3.52
  Type:   Direct-Access                    ANSI SCSI revision: 02
```

So, during the development I created two .sh scripts to make things easier for me.

```
$ cat backup_usb_writable_partition.sh
#!/bin/sh
mount /dev/sdb3 /mnt/usb
cp -r /mnt/usb/* usb/ --reply=y
umount /mnt/usb

$ cat update_flash_cramfs.sh
#!/bin/sh
mkcramfs rootfs/ images/esd_rootfs.cram
dd if=images/esd_rootfs.cram of=/dev/sdb2
```

Now format the USB Flash drive using:

```
$ fdisk /dev/sdb
```

Follow the on-screen menu to remove all current partitions and create three new partitions as the above table. Then formatting those partitions.

```
$ mke2fs /dev/sdb1
$ mke2fs /dev/sdb3
```

These will create `ext2` filesystem on partition 1 and 3. The first will be used for storing bootloader and kernel image and the third will be used for dynamic data. Then mount the first partition into a directory on the host system and make some directories as below:

```
$ mkdir /mnt
$ mkdir /mnt/usb
$ mount -t ext2 /dev/sdb1 /mnt/usb
$ mkdir /mnt/usb/boot
$ mkdir /mnt/usb/boot/grub
```

Boot Loader

First, load GRUB files from the host to the USB.

```
$ cp /boot/grub/stage1 /mnt/usb/boot/grub/
$ cp /boot/grub/stage2 /mnt/usb/boot/grub/
$ cp /boot/grub/e2fs_stage1_5 /mnt/usb/boot/grub/
```

Installing GRUB.

```
$ grub
grub> root (hd1,0)
grub> setup (hd1)
grub> quit
```

The above commands cannot be careless however. If you choose the wrong device such as (hd0,0) which is your hard drive instead of USB (hd1,0), there is an opportunity that you can destroy your host's bootloader. The `root (hd1,0)` command set GRUB to the USB partition that has `/boot` in order to load the kernel image. The setup command installs boot loader in the MBR

(Master Boot Record) of the USB at 0x80 address.

Edit a grub.conf

Edit a grub.conf in 1st partion.

```
$ vi /mnt/usb/boot/grub/grub.conf
```

`grub.conf` should contain something like this.

```
default=0
timeout=10
title Linusb 2.6.10 - patch usb
kernel /boot/vmlinuz-2.6.10-usb_patch root=/dev/sda2 rootdelay=6 init=startup.sh
```

There are a few things worth explaining here. First, I pointed grub to the that contain a Linux kernel image in the USB Flash's partition 1 (`/boot/vmlinuz-2.6.10-usb_patch`) and root file system in partition 2 (`root=/dev/sda2`.) Yes it is /dev/sda2 not /dev/sdb2 as the very first scsi device will be the USB Flash Drive after leaving this development environment.

`rootdelay=6` is to delay mounting of root filesystem for six seconds. I did this as I had discovered that the USB device partitioning was not ready right away after the kernel image is loaded therefore it would not be able to locate root file system image. The kernel would halt with kernel panic message on the screen. In order to make the kernel familiar with `rootdelay`. I needed a patch for Linux kernel version 2.6.10 (which of course will be included in 2.6.11 release) in which I will cover in the next section.

`init=startup.sh` is use to add a writable storage in the USB Flash Drive. I will discuss in Root File System section

Kernel

Download a kernel version from www.kernel.org. At the time of development the latest one is 2.6.10. Keep the kernel tar file in `~/usb_linux/kernel/` folder.

```
$ cd ~/usb_linux/kernel
$ tar xzvf linux-2.6.10.tar.gz
$ cd linux-2.6.10
```

As told earlier to make the kernel 2.6.10 support `rootdelay` boot option I needed a kernel patch below: [Appendix A]

```
--- linux-2.6.10/init/do_mounts.c.orig  2005-01-20 20:37:01.000000000 +0000
+++ linux-2.6.10/init/do_mounts.c       2005-01-20 20:44:47.190899080 +0000
@@ -6,6 +6,7 @@
 #include <linux/suspend.h>
 #include <linux/root_dev.h>
 #include <linux/security.h>

....
....

+
        rootflags=      [KNL] Set root filesystem mount option string

        rootfstype=     [KNL] Set root filesystem type
```

Saved the above lines to `~/usb_linux/kernel/linux-2.6.10/linux-2.6.10_usb.patch`. Then,

```
$ cd ~/usb_linux/kernel/linux-2.6.10
$ cp -p init/do_mounts.c init/do_mounts.c.orig
```

```
$ patch init/do_mounts.c linux-2.6.10_usb.patch
```

Change kernel configurations

```
$ make menuconfig
```

There are many configuration to choose from. To make a bootable USB Flash Drive, make sure these are compiled in:

Device Drivers --> SCSI device support
    --- SCSI support type (disk, tape, CD-ROM)
    <*> SCSI disk support
Device Drivers --> USB support
    <*> Support for Host-side USB
    <*> OHCI HCD support
    <*> UHCI HCD (most Intel and VIA) support

    <*> USB Mass Storage support

File systems --> Miscellaneous filesystems
    <*> Compressed ROM file system support (cramfs)

One may configure the kernel to suit any target. The `diff` of my `.config` from the original one (the one that comes in as a standard configuration) is

```
CONFIG_EMBEDDED=y
CONFIG_MTD=y
CONFIG_MTD_CHAR=y
CONFIG_MTD_BLOCK=y
CONFIG_MTD_CFI=y
CONFIG_MTD_GEN_PROBE=y
CONFIG_MTD_CFI_AMDSTD=y
CONFIG_MTD_CFI_UTIL=y
CONFIG_SCSI=y
CONFIG_BLK_DEV_SD=y
CONFIG_SCSI_QLA2XXX=y
CONFIG_MII=y
CONFIG_E100=y
CONFIG_E1000=y
CONFIG_USB_OHCI_HCD=y
CONFIG_USB_UHCI_HCD=y
CONFIG_USB_STORAGE=y
CONFIG_ROMFS_FS=y
CONFIG_JFFS_FS=m
CONFIG_JFFS_FS_VERBOSE=0
CONFIG_JFFS2_FS=y
CONFIG_CRAMFS=y
CONFIG_ZLIB_DEFLATE=y
```

And make sure that the kernel supports for Unix98 Pts (/dev/pts), It should come default.
Now it is time to compile the kernel and copy it over to the USB divice.

```
$ make dep
$ make bzImage
$ cp arch/i386/boot/bzImage /mnt/usb/boot/vmlinuz-2.6.10-usb_patch
$ umount /mnt/usb
```

Root File System

The file system which will be place in 2nd and 3rd partition of USB Flash Drive consists of:

1. Read-only cramfs file systems in 2nd partition
2. Writable ext2 storage in 3rd partition which contains dynamic data such as `etc/`, `home/`, `root/`, `var/` folders.

I created some directories and some files in root file system folder below

```
$ cd ~/usb_linux/rootfs
$ ls -l
total 48
drwxr-xr-x  2 root root 4096 Mar 16 11:34 bin
-rwxr--r--  1 root root  168 Mar 18 14:57 chroot.script
-rwxr--r--  1 root root   53 Mar 18 09:21 chroot.sh
drwxr-xr-x  3 root root 4096 Mar 18 09:20 dev
lrwxrwxrwx  1 root root    7 Mar 10 11:24 etc -> usb/etc
lrwxrwxrwx  1 root root    8 Mar 10 11:24 home -> usb/home
drwxr-xr-x  2 root root 4096 Mar 15 14:47 lib
lrwxrwxrwx  1 root root   11 Mar 16 11:34 linuxrc -> bin/busybox
drwxr-xr-x  2 root root 4096 Feb 23 10:25 mnt
drwxr-xr-x  2 root root 4096 Feb 23 10:25 proc
lrwxrwxrwx  1 root root    8 Mar 10 11:25 root -> usb/root
drwxr-xr-x  2 root root 4096 Mar 16 11:34 sbin
-rwxr--r--  1 root root   60 Mar 10 09:48 startup.sh
drwxr-xr-x  2 root root 4096 Feb 23 10:25 tmp
drwxr-xr-x  2 root root 4096 Mar  4 13:54 usb
drwxr-xr-x  6 root root 4096 Mar 18 09:13 usr
lrwxrwxrwx  1 root root    7 Mar 10 11:24 var -> usb/var
```

`Startup.sh` is a startup script. It will be called right after kernel loaded as specify as a boot option in GRUB instead of the default `init` process. This script makes sure that the `etc/` directory and others are available before system initialization. The `exec` command is forced the process to use the same `pid` instead of fork another process as `init` must be pid 0. Contents of the script:

```
$ cat startup.sh
#!/bin/sh
/bin/mount -t ext2 /dev/sda3 /usb
exec /sbin/init
```

`I` have also created some `chroot` scripts used during the development. Here they are:

```
$ cat chroot.sh
#!/bin/sh
DEVDIR=`pwd`
chroot $DEVDIR /chroot.script

$ cat chroot.script
#!/bin/sh
PS1="\h\w:\\$ "
export PS1
mount proc /proc -t proc
mount /dev/sdb3 /usb
/bin/ash
echo "Exit Build root........Bye"
umount /usb
umount /proc
```

To populate file system in `/dev` use `mknod`.

```
$ cd dev/
```

```
$ mknod -m 600 mem c 1 1
$ mknod -m 666 null c 1 3
...
$ mknod -m 600 ram0 b 1 0
$ mknod -m 644 sda b 8 0
...
$ ln -s /proc/self/fd fd
$ ln -s fd/0 stdin
$ ln -s fd/1 stdout
$ ln -s fd/2 stderr
$ cd ..
```

Keep op populating these special devices and symbolic links above and the content of `dev/` should look like this:

```
$ ls -l dev/
total 4
crw-------  1 root root 5,  1 Feb 22 15:32 console
lrwxrwxrwx  1 root root    13 Feb 23 11:18 fd -> /proc/self/fd
crw-------  1 root root 1,  1 Feb 22 15:32 mem
crw-rw-rw-  1 root root 1,  3 Feb 22 15:33 null
crw-rw-rw-  1 root root 5,  2 Mar 10 10:52 ptmx
drwxr-xr-x  2 root root  4096 Mar 10 09:35 pts
brw-------  1 root root 1,  0 Feb 22 15:32 ram0
brw-rw----  1 root root 1,  1 Feb 22 15:32 ram1
crw-r--r--  1 root root 1,  8 Feb 22 15:33 random
brw-r--r--  1 root root 8,  0 Mar  4 14:10 sda
brw-r--r--  1 root root 8,  1 Mar  4 14:11 sda1
brw-r--r--  1 root root 8,  2 Mar  4 14:11 sda2
brw-r--r--  1 root root 8,  3 Mar  4 14:11 sda3
brw-r--r--  1 root root 8,  4 Mar 15 14:07 sda4
brw-r--r--  1 root root 8,  5 Mar 15 14:06 sda5
brw-r--r--  1 root root 8, 16 Mar 15 14:05 sdb
brw-r--r--  1 root root 8, 17 Mar 15 14:05 sdb1
brw-r--r--  1 root root 8, 18 Mar 15 14:06 sdb2
brw-r--r--  1 root root 8, 19 Mar 15 14:06 sdb3
brw-r--r--  1 root root 8, 20 Mar 15 14:06 sdb4
brw-r--r--  1 root root 8, 21 Mar 15 14:07 sdb5
lrwxrwxrwx  1 root root     4 Feb 23 11:19 stderr -> fd/2
lrwxrwxrwx  1 root root     4 Feb 23 11:18 stdin -> fd/0
lrwxrwxrwx  1 root root     4 Feb 23 11:19 stdout -> fd/1
crw-rw-rw-  1 root root 5,  0 Feb 22 15:32 tty
crw-------  1 root root 4,  0 Feb 22 15:33 tty0
crw-------  1 root root 4,  1 Mar  8 14:06 tty1
crw-------  1 root root 4,  2 Mar  8 16:40 tty2
crw-------  1 root root 4,  3 Mar  8 16:40 tty3
crw-------  1 root root 4, 64 Feb 22 15:33 ttyS0
crw-------  1 root root 4, 65 Feb 22 15:33 ttyS1
crw-r--r--  1 root root 1,  9 Mar 22 09:30 urandom
crw-rw-rw-  1 root root 1,  5 Feb 22 15:33 zero
```

Note: that `ptmx, pts` devices are needed for `telnetd, urandom` is need for `udhcpc and tty` devices are needed for virtual consoles.

To install Busybox, download `busybox-1.00.tar.gz` from http://www.busybox.net/ into `~/usb_linux/sysapps/` then modify the configuration file, compile it and copy the binary to `rootfs` folder.

```
$ cd ~/usb_linux/sysapps
$ tar xzvf busybox-1.00.tar.gz
$ cd busybox-1.00
$ make menuconfig
$ make install
$ cp -r _install/* ~/usb_linux/rootfs –-reply=y
```

I built Busybox with static link option along with other tools. The `diff` of `.config` of my Busybox `.config` and the original that comes with the package without any modification is:

```
CONFIG_FEATURE_SUID=y
CONFIG_FEATURE_SUID_CONFIG=y
CONFIG_STATIC=y
CONFIG_FEATURE_FANCY_HEAD=y
CONFIG_HOSTID=y
CONFIG_LENGTH=y
CONFIG_OD=y
CONFIG_PRINTF=y
CONFIG_REALPATH=y
CONFIG_WHO=y
CONFIG_AWK=y
CONFIG_FEATURE_AWK_MATH=y
CONFIG_PATCH=y
CONFIG_USE_BB_PWD_GRP=y
CONFIG_ADDGROUP=y
CONFIG_DELGROUP=y
CONFIG_ADDUSER=y
CONFIG_DELUSER=y
CONFIG_GETTY=y
CONFIG_FEATURE_U_W_TMP=y
CONFIG_LOGIN=y
CONFIG_FEATURE_SECURETTY=y
CONFIG_PASSWD=y
CONFIG_SU=y
CONFIG_SULOGIN=y
CONFIG_VLOCK=y
CONFIG_CROND=y
CONFIG_CRONTAB=y
CONFIG_ARPING=y
CONFIG_FTPGET=y
CONFIG_FTPPUT=y
CONFIG_HTTPD=y
CONFIG_FEATURE_HTTPD_BASIC_AUTH=y
CONFIG_FEATURE_HTTPD_RELOAD_CONFIG_SIGHUP=y
CONFIG_FEATURE_HTTPD_CGI=y
CONFIG_FEATURE_HTTPD_ENCODE_URL_STR=y
CONFIG_IFUPDOWN=y
CONFIG_FEATURE_IFUPDOWN_IP_BUILTIN=y
CONFIG_FEATURE_IFUPDOWN_IPV4=y
CONFIG_INETD=y
CONFIG_FEATURE_INETD_SUPPORT_BILTIN_ECHO=y
CONFIG_FEATURE_INETD_SUPPORT_BILTIN_DISCARD=y
CONFIG_FEATURE_INETD_SUPPORT_BILTIN_TIME=y
CONFIG_FEATURE_INETD_SUPPORT_BILTIN_DAYTIME=y
CONFIG_FEATURE_INETD_SUPPORT_BILTIN_CHARGEN=y
CONFIG_IP=y
CONFIG_FEATURE_IP_ADDRESS=y
CONFIG_FEATURE_IP_LINK=y
CONFIG_FEATURE_IP_ROUTE=y
CONFIG_NC=y
CONFIG_NETSTAT=y
CONFIG_NSLOOKUP=y
CONFIG_TELNET=y
CONFIG_FEATURE_TELNET_TTYPE=y
CONFIG_FEATURE_TELNET_AUTOLOGIN=y
CONFIG_TELNETD=y
CONFIG_TFTP=y
CONFIG_FEATURE_TFTP_GET=y
CONFIG_FEATURE_TFTP_PUT=y
CONFIG_TRACEROUTE=y
CONFIG_UDHCPC=y
CONFIG_TOP=y
FEATURE_CPU_USAGE_PERCENTAGE=y
CONFIG_ASH_GETOPTS=y
CONFIG_ASH_CMDCMD=y
CONFIG_FEATURE_COMMAND_HISTORY=30
CONFIG_GETOPT=y
```

```
CONFIG_RDATE=y
CONFIG_NFSMOUNT=y
```

Then, another important thing we need to do is the set permission of `bin/busybox` to run busybox applets with user/group id of the file owner instead of originator otherwise you may get message from busybox when trying to run its applets like *"cannot set groups: Operation not permit"* especially `su` command. To enable this:

```
$ cd ~/usb_linux
$ chmod 6111 rootfs/bin/busybox
$ ls -l rootfs/bin/busybox
---s--s--x    1 roo     root       946484 Mar 16 04:34 busybox
```

Now we should be able to `chroot` which will change the root directory to `rootfs`.

```
$ ./chroot.sh
```

We need some configuration files in `etc/` like `nsswitch.conf`, `resolv.conf`, `ssh_config` and `sshd_config` from the host. In addition, edit these files as below: (> denotes `chroot` prompt)

```
> cat etc/busybox.conf
[SUID]
su = ssx root.root

> cat etc/fstab
/dev/sda2       /               cramfs          defaults        1       1
none            /proc           proc            defaults        0       0
none            /dev/pts        devpts          gid=5,mode=620  0       0

> cat etc/group
root:x:0:

> cat etc/passwd
root:$1$$UOV6ZlYPVr7i5kgPyhPM50:0:0:Super User:/root:/bin/sh
sshd:x:103:99:Operator:/var:/bin/sh

> cat etc/network.conf
DEVICE=eth0
# DHCP if not used, comment out
#DHCP=yes
IPADDR=192.168.1.1
NETMASK=255.255.255.0
GATEWAY=192.168.1.100
BCAST=192.168.1.255
HOSTNAME=usblinux
PRIDNS=xxx.xxx.xxx.xxx
SECDNS=
```

`httpd.conf` contains configurations for busybox built-in `httpd`. See busybox's `httpd.c` for more details.

```
> cat etc/httpd.conf
# Config file for busybox httpd
# Allow any IP address to connect
A:*
# Allow admin user to connect to the base directory
/cgi-bin:*
/:admin:admin
```

At the system initialization `inittab` is called by `init`. We use `ttyx::askfirst::/bin/login` for any virtual terminal access to the system. Details:

```
> cat etc/inittab
# Startup the system
null::sysinit:/bin/mount -o remount,rw /
null::sysinit:/bin/mount -t proc proc /proc
```

```
null::sysinit:/bin/mount -t devpts devpts /dev/pts
null::sysinit:/bin/mount -a
null::sysinit:/bin/hostname -F /etc/hostname
null::sysinit:/sbin/ifconfig lo 127.0.0.1 up
null::sysinit:/sbin/route add -net 127.0.0.0 netmask 255.0.0.0 lo
# now run any rc scripts
::sysinit:/etc/init.d/rcS

# Set up a couple of getty's
#tty1::respawn:/sbin/getty 38400 tty1
#tty2::respawn:/sbin/getty 38400 tty2
tty1::askfirst:/bin/login
tty2::askfirst:/bin/login
tty3::askfirst:/bin/login

# Put a getty on the serial port
#ttyS0::respawn:/sbin/getty -L ttyS0 115200 vt100

# Logging junk
#null::sysinit:/bin/touch /var/log/messages
#null::respawn:/sbin/syslogd -n -m 0
#null::respawn:/sbin/klogd -n
#tty3::respawn:/usr/bin/tail -f /var/log/messages

# Stuff to do for the 3-finger salute
::ctrlaltdel:/sbin/reboot

# Stuff to do before rebooting
#null::shutdown:/usr/bin/killall klogd
#null::shutdown:/usr/bin/killall syslogd
null::shutdown:/bin/umount -a -r
null::shutdown:/sbin/swapoff -a
```

Now we need to edit rcS script and its sidekicks in `etc/init.d/`. I used conventional S<number><process> as a filename for processes and services that are required to be available at the boot-up numerically.

```
> ls -l etc/init.d/
total 24
-rwxr--r--  1 root root 1432 Mar 21 15:51 functions
-rwxr--r--  1 root root  911 Mar 21 15:51 rcS
-rwxr--r--  1 root root 1164 Mar 21 15:51 S20urandom
-rwxr--r--  1 root root 2776 Mar 21 15:51 S40network
-rwxr--r--  1 root root 1146 Mar 21 15:51 S50sshd
-rwxr--r--  1 root root  464 Mar 21 15:51 S60httpd
```

See [Appendix A] for contents of files.

To be using busybox DHCP client, `udhcpc`, we need `default.script` in `/usr/share/udhcpc/` (`mkdir` if it is not available.)

```
> exit
$ vi ~/usb_linux/rootfs/usr/share/udhcpc/default.script
```

See [Appendix A] for detail. As a matter of fact this file is included in a small Linux Firewall system on a diskette called `coyote-2.21`. I downloaded it from http://www.coyotelinux.com into `~/usb_linux/sysapps/coyote-2.21-build.tar.bz2`. Then

```
$ cd ~/usb_linux/sysapps
$ tar xjvf coyote-2.21-build.tar.bz2
$ cp -p coyote-2.21-build/usr/share/udhcpc/default.script \
 ~/usb_linux/rootfs/usr/share/udhcpc/default.script
```

I also used a complied binaries of `sshd`, `ssh` in coyotelinux package.

```
$ cp -p coyote-2.21-build/usr/bin/ssh* ~/usb_linux/rootfs/usr/bin
```

```
$ cp -p coyote-2.21-build/usr/sbin/ssh* ~/usb_linux/rootfs/usr/sbin
```

As coyotelinux is built against `uClibc` with dynamic link then we need to copy its `lib` to our root file system as listed below. And also some networking tools like busybox's `nslookup` need libraries even though busybox is static link built. These lib are available on the host machine or in coyotelinux's `lib` folder.

```
$ cd ../rootfs
$ ls -l lib/
total 2516
-rwxr-xr-x  1 root root  106892 Mar 11 15:35 ld-linux.so.2
-rwxr-xr-x  1 root root   16328 Feb 22  2004 ld-uClibc.so.0
-rw-r--r--  1 root root    9036 Mar 15 14:44 libcrypt-0.9.26.so
lrwxrwxrwx  1 root root      18 Mar 15 14:44 libcrypt.so.0 -> libcrypt-0.9.26.so
lrwxrwxrwx  1 root root      19 Mar 15 14:28 libc.so.0 -> libuClibc-0.9.26.so
-rwxr-xr-x  1 root root 1443920 Mar 11 13:52 libc.so.6
-rw-r--r--  1 root root    5524 Mar 15 14:47 libdl-0.9.26.so
lrwxrwxrwx  1 root root      15 Mar 15 14:47 libdl.so.0 -> libdl-0.9.26.so
-rwxr-xr-x  1 root root   22172 Mar 11 13:51 libnss_dns-2.3.3.so
-rwxr-xr-x  1 root root   17804 Mar 11 13:51 libnss_dns.so.1
-rwxr-xr-x  1 root root   22172 Mar 11 13:51 libnss_dns.so.2
-rwxr-xr-x  1 root root   50944 Mar 11 13:51 libnss_files-2.3.3.so
-rwxr-xr-x  1 root root   50440 Mar 11 13:51 libnss_files.so.1
-rwxr-xr-x  1 root root   50944 Mar 11 13:51 libnss_files.so.2
-rwxr-xr-x  1 root root   43528 Mar 11 15:22 libnss_nis-2.3.3.so
-rwxr-xr-x  1 root root   51772 Mar 11 15:22 libnss_nisplus-2.3.3.so
-rwxr-xr-x  1 root root   51772 Mar 11 15:22 libnss_nisplus.so.2
-rwxr-xr-x  1 root root   39336 Mar 11 15:22 libnss_nis.so.1
-rwxr-xr-x  1 root root   43528 Mar 11 15:22 libnss_nis.so.2
-rwxr-xr-x  1 root root   76588 Mar 11 13:52 libresolv-2.3.3.so
-rwxr-xr-x  1 root root   76588 Mar 11 13:52 libresolv.so.2
-rw-r--r--  1 root root  263004 Feb 22  2004 libuClibc-0.9.26.so
-rw-r--r--  1 root root    4208 Mar 15 14:38 libutil-0.9.26.so
lrwxrwxrwx  1 root root      17 Mar 15 14:38 libutil.so.0 -> libutil-0.9.26.so
lrwxrwxrwx  1 root root      13 Mar 15 14:38 libz.so.1 -> libz.so.1.1.4
-rw-r--r--  1 root root   44716 Mar 15 14:38 libz.so.1.1.4

$ ls -l usr/lib/
total 2652
-rw-r--r--  1 root root   10806 Mar 15 14:44 libcrypt.a
-rw-r--r--  1 root root 1703098 Mar 15 14:44 libcrypto.a
lrwxrwxrwx  1 root root      18 Mar 15 14:44 libcrypto.so -> libcrypto.so.0.9.7
lrwxrwxrwx  1 root root      18 Mar 15 14:44 libcrypto.so.0 -> libcrypto.so.0.9.7
-rw-r--r--  1 root root  921980 Mar 15 14:44 libcrypto.so.0.9.7
lrwxrwxrwx  1 root root      10 Mar 15 14:44 libcrypt_pic.a -> libcrypt.a
lrwxrwxrwx  1 root root      18 Mar 15 14:44 libcrypt.so -> /lib/libcrypt.so.0
lrwxrwxrwx  1 root root      15 Mar 15 14:46 libdl.so -> /lib/libdl.so.0
-rw-r--r--  1 root root   60408 Mar 15 14:43 libz.a
lrwxrwxrwx  1 root root      18 Mar 15 14:43 libz.so -> /lib/libz.so.1.1.4
```

Now all the root file system and its configurations are all set. We need to make a `cramfs` image of `rootfs` and put it into USB Flash drive and backup the content of writable storage in `/dev/sdb3` into the host's `~/usb_linux/usb`.

```
$ cd ~/usb_linux
$ ./update_flash_cramfs.sh
$ ./backup_usb_writable_partition.sh
```

Now you are ready to boot a machine with this USB Flash Drive plugged in and greeted by a login prompt and have all the services of `httpd, sshd`.

```
Please press any key to activate this consle.
login: root
password:
BusyBox v1.00 (2005.02.22-08:36+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.
```

usblinux:#

Appendix A :: Files contents

linux-2.6.10_usb.patch

```
--- linux-2.6.10/init/do_mounts.c.orig  2005-01-20 20:37:01.000000000 +0000
+++ linux-2.6.10/init/do_mounts.c       2005-01-20 20:44:47.190899080 +0000
@@ -6,6 +6,7 @@
 #include <linux/suspend.h>
 #include <linux/root_dev.h>
 #include <linux/security.h>
+#include <linux/delay.h>

 #include <linux/nfs_fs.h>
 #include <linux/nfs_fs_sb.h>
@@ -228,8 +229,16 @@
        return 1;
 }

+static unsigned int __initdata root_delay;
+static int __init root_delay_setup(char *str)
+{
+       root_delay = simple_strtoul(str, NULL, 0);
+       return 1;
+}
+
 __setup("rootflags=", root_data_setup);
 __setup("rootfstype=", fs_names_setup);
+__setup("rootdelay=", root_delay_setup);

 static void __init get_fs_names(char *page)
 {
@@ -387,6 +396,12 @@

        mount_devfs();

+       if (root_delay) {
+               printk(KERN_INFO "Waiting %dsec before mounting root device...\n",
+                      root_delay);
+               ssleep(root_delay);
+       }
+
        md_run_setup();

        if (saved_root_name[0]) {
--- linux-2.6.10/Documentation/kernel-parameters.txt.orig       2005-01-21
17:18:20.000000000 +0000
+++ linux-2.6.10/Documentation/kernel-parameters.txt    2005-01-21
17:22:29.000000000 +0000
@@ -1072,6 +1072,9 @@ running once the system is up.

        root=           [KNL] Root filesystem

+       rootdelay=      [KNL] Delay (in seconds) to pause before attempting to
+                       mount the root filesystem
+
        rootflags=      [KNL] Set root filesystem mount option string

        rootfstype=     [KNL] Set root filesystem type
```

rcS

```
#!/bin/sh
# Start all init scripts in /etc/init.d
# executing them in numerical order.
#
for i in /etc/init.d/S??* ;do

     # Ignore dangling symlinks (if any).
     [ ! -f "$i" ] && continue

     case "$i" in
        *.sh)
     # Source shell script for speed.
     (
        trap - INT QUIT TSTP
        set start
        . $i
     )
     ;;
         *)
     # No sh extension, so fork subprocess.
     $i start
     ;;
     esac
done
```

functions

```
#!/bin/sh
# First set up a default search path.
export PATH="/sbin:/usr/sbin:/bin:/usr/bin"
# Get a sane screen width
BOOTUP=color
RES_COL=60
MOVE_TO_COL="echo -en \\033[${RES_COL}G"
SETCOLOR_SUCCESS="echo -en \\033[1;32m"
SETCOLOR_FAILURE="echo -en \\033[1;31m"
SETCOLOR_WARNING="echo -en \\033[1;33m"
SETCOLOR_NORMAL="echo -en \\033[0;39m"
LOGLEVEL=1
echo_success() {
  [ "$BOOTUP" = "color" ] && $MOVE_TO_COL
  echo -n "[  "
  [ "$BOOTUP" = "color" ] && $SETCOLOR_SUCCESS
  echo -n "OK"
  [ "$BOOTUP" = "color" ] && $SETCOLOR_NORMAL
  echo -n "  ]"
  echo -ne "\r"
  return 0
}
echo_failure() {
  [ "$BOOTUP" = "color" ] && $MOVE_TO_COL
  echo -n "["
  [ "$BOOTUP" = "color" ] && $SETCOLOR_FAILURE
  echo -n "FAILED"
  [ "$BOOTUP" = "color" ] && $SETCOLOR_NORMAL
  echo -n "]"
  echo -ne "\r"
  return 1
}
# Log that something succeeded
success() {
  [ "$BOOTUP" != "verbose" ] && echo_success
}
# Log that something failed
failure() {
  [ "$BOOTUP" != "verbose" -a -z "$LSB" ] && echo_failure
}
```

S20urandom

```
#! /bin/sh
#
# urandom       This script saves the random seed between reboots.
#               It is called from the boot, halt and reboot scripts.
#
# Version:      @(#)urandom  1.33  22-Jun-1998  miquels@cistron.nl
#

[ -c /dev/urandom ] || exit 0
#. /etc/default/rcS
. /etc/init.d/functions

case "$1" in
        start|"")
                if [ "$VERBOSE" != no ]
                then
                        echo -n "Initializing random number generator... "
                fi
                # Load and then save 512 bytes,
                # which is the size of the entropy pool
                if [ -f /etc/random-seed ]
                then
                        cat /etc/random-seed >/dev/urandom
                fi
                rm -f /etc/random-seed
                umask 077
                dd if=/dev/urandom of=/etc/random-seed count=1 \
                        >/dev/null 2>&1 || echo "urandom start: failed."
                umask 022
                [ "$VERBOSE" != no ] && success #echo "done."
                echo
                ;;
        stop)
                # Carry a random seed from shut-down to start-up;
                # see documentation in linux/drivers/char/random.c
                [ "$VERBOSE" != no ] && echo -n "Saving random seed... "
                umask 077
                dd if=/dev/urandom of=/etc/random-seed count=1 \
                        >/dev/null 2>&1 || echo "urandom stop: failed."
                [ "$VERBOSE" != no ] && success #echo "done."
                echo
                ;;
        *)
                echo "Usage: urandom {start|stop}" >&2
                exit 1
                ;;
esac
```

S40network

```
#!/bin/sh
network_start () {

export PATH="/bin:/usr/bin:/usr/sbin:/sbin"

. /etc/init.d/functions

echo -n "Checking Network Configuration: "
if [ ! -f /etc/network.conf ]; then
  failure
  echo
  echo "Network configuration file (network.conf) not found."
  echo "Please check if the network.conf is found in the USB flash drive"
  sleep 1800
  halt
else
  success
  echo
fi

chmod +x /etc/network.conf
. /etc/network.conf

test -x /sbin/ifconfig || halt

if [ -z ${DHCP} ] ; then
  echo -n "Loading Static Network Interface: "
  /sbin/ifconfig ${DEVICE} ${IPADDR} netmask ${NETMASK} broadcast ${BCAST}
  /sbin/route del default 2>/dev/null
  /sbin/route add default gw ${GATEWAY}
  /bin/hostname ${HOSTNAME}
  success
  echo "nameserver ${PRIDNS}" > /etc/resolv.conf
  echo "nameserver ${SECDNS}" >> /etc/resolv.conf
  echo
else
  echo -n "Loading DHCP Network Interface: "
  if [ -x /sbin/udhcpc ] ; then
     /sbin/udhcpc -i ${DEVICE} >/dev/null 2>/dev/null && success || failure
     #success
     echo
  else
     failure
     echo
  fi
fi

echo -n "Checking Network Connection: "
TEST=`/sbin/ifconfig | grep ${DEVICE}`
if [ -z "${TEST}" ]; then
  failure
  echo
  echo "Unable to detect network device."
  echo "The network interface was not loaded properly."
  TEST=`cat /proc/net/dev| grep eth0`

  if [ -z ${TEST} ]; then
    echo "Network interface card not supported."
    echo "It is possible you are using an unsupported NIC."
  else
    echo "Check your configuration in the network.conf file."
  fi
  sleep 1800
  halt
else
  success
```

```
  echo
fi

#if test -z "\$VNCHOST"
#then
#  if test -z "\$TSHOST"
#  then
#    echo "Both VNCHOST and TSHOST are not set."
#    echo "Press any key to continue ..."
#    read DUMP
#    /bin/sh
#  else
#    rdesktop \$TSHOST
#  fi
#else
#  svncviewer \$VNCHOST:0
#fi
}

start() {
        echo "Starting network..."
        #/sbin/ifconfig ${DEVICE} up
        network_start
}
stop() {
        . /etc/network.conf
        . /etc/init.d/functions
        echo -n "Stopping network..."
        /sbin/ifconfig ${DEVICE} down && success || failure
        #success
        killall -q udhcpc
        echo
}
restart() {
        stop
        start
}

case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  restart|reload)
        restart
        ;;
  *)
        echo $"Usage: $0 {start|stop|restart}"
        exit 1
esac

exit $?
```

S50sshd

```
#!/bin/sh
#
# sshd        Starts sshd.
#

. /etc/init.d/functions

# Make sure the ssh-keygen progam exists
[ -f /usr/bin/ssh-keygen ] || exit 0

# Check for the SSH1 RSA key
if [ ! -f /etc/ssh_host_key ] ; then
        echo Generating RSA Key...
        /usr/bin/ssh-keygen -t rsa1 -f /etc/ssh_host_key -C '' -N ''
fi

# Check for the SSH2 RSA key
if [ ! -f /etc/ssh_host_rsa_key ] ; then
        echo Generating RSA Key...
        /usr/bin/ssh-keygen -t rsa -f /etc/ssh_host_rsa_key -C '' -N ''
fi

# Check for the SSH2 DSA key
if [ ! -f /etc/ssh_host_dsa_key ] ; then
        echo Generating DSA Key...
        echo THIS CAN TAKE A MINUTE OR TWO DEPENDING ON YOUR PROCESSOR!
        echo
        /usr/bin/ssh-keygen -t dsa -f /etc/ssh_host_dsa_key -C '' -N ''
fi

umask 077

start() {
        echo -n "Starting sshd: "
        /usr/sbin/sshd && success || failure
        touch /var/lock/sshd
        echo
}
stop() {
        echo -n "Stopping sshd: "
        killall sshd && success || failure
        rm -f /var/lock/sshd
        echo
}
restart() {
        stop
        start
}

case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  restart|reload)
        restart
        ;;
  *)
        echo $"Usage: $0 {start|stop|restart}"
        exit 1
esac

exit $?
```

S60httpd

```
        #!/bin/sh
        #
        # Start httpd
        #

        . /etc/init.d/functions

        start() {
                echo -n "Starting httpd..."
                /usr/sbin/httpd -c /etc/httpd.conf -h /var/www  -r "Web Interface
Configuration" \
                && success || failure
                echo
        }
        stop() {
                echo -n "Stopping httpd..."
                killall httpd && success || failure
                echo
        }
        restart() {
                stop
                start
        }

        case "$1" in
                start)
                        start
                        ;;
                stop)
                        stop
                        ;;
                restart)
                        restart
                        ;;
                *)
                        echo "Usage: $0 {start|stop|restart}"
                        exit 1
        esac

        exit $?
```

default.script

```
#!/bin/sh

# udhcpc script edited by Tim Riker <Tim@Rikers.org>

[ -z "$1" ] && echo "Error: should be called from udhcpc" && exit 1

RESOLV_CONF="/etc/resolv.conf"
[ -n "$broadcast" ] && BROADCAST="broadcast $broadcast"
[ -n "$subnet" ] && NETMASK="netmask $subnet"

case "$1" in
        deconfig)
                /sbin/ifconfig $interface 0.0.0.0
                ;;

        renew|bound)
                /sbin/ifconfig $interface $ip $BROADCAST $NETMASK

                if [ -n "$router" ] ; then
                        echo "deleting routers"
                        while route del default gw 0.0.0.0 dev $interface ; do
                                :
                        done

                        for i in $router ; do
                                route add default gw $i dev $interface
                        done
                fi

                echo -n > $RESOLV_CONF
                [ -n "$domain" ] && echo search $domain >> $RESOLV_CONF
                for i in $dns ; do
                        echo adding dns $i
                        echo nameserver $i >> $RESOLV_CONF
                done
                ;;
esac

exit 0
```