Embedding Redhat Linux in a DiskOnChip - HOWTO

Don Davies, Prosig Ltd (don.davies@prosig.com)
October 2002

Describes the configuration and setup of a development environment for a Single Board Computer running Redhat Linux from a DiskOnChip device.

Contents

1.0 Introduction

Central to the development of the PROTOR distributed vibration monitoring system is the setup and configuration of an embedded single board computer for remote monitoring applications. The embedded processor sits within a special purpose chassis along with data acquisition and conditioning modules. External communication is via Ethernet and internal via both PC-104 and USB. These units may be installed in potentially harsh environments and so a system which supports solid-state storage rather than a mechanical device is required. These units also run Linux as this provides the most reliable, best cost-performance platform for a remote monitoring applications.

This note describes the setup and configuration of a development system for a suitable single board computer which supports the solid-state DiskOnChip device from M-Systems Inc. The note also shows how to access the DiskOnChip from within a Linux environment and how to make the system bootable from DiskOnChip so that no hard disk is required. In order to run Linux from a DiskOnChip device some details are provided on the essential files needed from a standard Redhat Linux distribution, together with some useful utilities which allow a fully functional, small-footprint Linux system to be stored and run from a 32MB DiskOnChip device.

1.1 Hardware Details

The single board computer chosen as the embedded processor for these units is the Nova 7892 card by ICP Electronics Inc (www.icpacquire.com). This card meets all the general requirements of an SBC for use within PROTOR. Other cards with similar facilities are available from other manufacturers.

• Footprint	146mm x 203mm (5.25" footprint)
• CPU	Supports Socket 370 Celeron and Pentium III CPUs
• Memory	Supports one 168-pin DIMM socket (upto 256Mbytes)
• Ethernet	On-board Intel 82559 10/100 Mbps interface with RJ45 connector.

• PC-104	Supports PC-104 expansion
• USB	Supports 2 USB Ports
• Solid-state storage	Supports M-System DiskOnChip

For the PROTOR application the NOVA-7892 card has been populated with an 866MHz Pentium III CPU and 64Mbytes of memory. For initial development a standard IDE hard disk is used. The eventual embedded application code is to run under Linux from DiskOnChip.

A general development system for the embedded system was assembled which consisted of :

- Nova 7892 SBC
- Pentium III CPU
- 64Mbytes memory
- 4Gbyte IDE Disk
- 32Mbyte DiskOnChip
- 1.4Mbyte floppy drive
- CD-ROM drive

1.2 System Configuration

In order for the DiskOnChip device to be recognised by the system BIOS and therefore useable within this environment it must firstly be initialised and formatted. Unfortunately the tools necessary to perform this action are DOS based. Therefore, for this development environment, the hard disk for the system is best partitioned with a small 32Mbyte partition was reserved at the beginning of the disk for DOS. The rest of the disk was reserved for the Linux system and Linux swap partition. The following is an example configuration.

```
Disk /dev/hda: 255 heads, 63 sectors, 524 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1             5       495   3911827+  83  Linux
/dev/hda2   *         1         4     32098+   4  FAT16 <32M
/dev/hda3           496       524    265072+  82  Linux swap
```

The system was initially booted from a standard DOS boot floppy. The hard disk partitioned and the initial DOS partition (C:\) formatted and made bootable.

2.0 DOS Development Environment

As mentioned in the previous section a simple DOS development environment is required for the initialisation and formatting of the DiskOnChip device. This environment is only required for the initial setup of the device. Once the DiskOnChip has been formatted, Linux loaded and made bootable then DOS is no longer required. The following describes the tools needed for the DOS environment in order to simplify the DiskOnChip development.

2.1 DiskOnChip Tools

In order to initialise and format a new DiskOnChip device a set of tools provided by the DiskOnChip manufacturers, M-Systems must be obtained. These tools (called TrueFFS or True Flash File System) are DOS utilities and must be downloaded from the web site http://www.m-sys.com/content/Developer/DOS.asp and installed into the DOS partition provided on the development system for this purpose.

Of the software provided my M-Systems, DFORMAT is probably the main tool required to get started with the DiskOnChip. DFORMAT initializes a "virgin" DiskOnChip and prepares it for the application's needs. DFORMAT provides the following.

- Bad block scanning
- Partitioning the DiskOnChip to disk and binary partitions
- Setting hardware protection configurations

- BIOS related operations
- Boot replacement operations

Note. M-Systems provide two versions of these tools (the latest release v5.0.4 and an older release v4.2). The following sections describe the configuration of Linux using MTD device support. Currently this software only works with the older (V4.2) TrueFFS format utility and so the newer (v5.0.4) should not be used.

You can try the format of the DiskOnChip device with the following command

```
    DFORMAT /WIN:D600 /S:DOC42.EXB
```

Note this command is for the v4.2 toolset. If a higher version is used then the EXB filename will need to change accordingly.

If the device is recognised then the format utility will show the capacity and show progress as the device is formatted. Following the format the DiskOnChip device should be acccessible as device D:\ from DOS.

2.2 Boot Loader

In order to simplify the development and booting of Linux kernels and to allow simple co-existence with DOS then the widely available DOS Linux loader module LOADLIN is recommended to be downloaded and installed within the DOS partition. Loadlin is a simple DOS based utility that loads the Linux kernel into memory from DOS. This allows a number of development Linux kernels to be built , copied to the DOS partition and loaded. Loadlin may be downloaded from ftp://ftp.linux.sh/pub/loadlin.exe.

2.3 MS-DOS System Startup

To facilitate DiskOnChip development the AUTOEXEC.BAT and CONFIG.SYS files within the DOS partition may be customised to allow various utilities and boot options to be readily available from the menu presented at boot time. The following are typical entries which may be added to any existing files.

CONFIG.SYS

```
. . .
[MENU]
MENUITEM=LLOCAL,Boot Linux 2.4.7  kernel(MTD) root on /dev/hda1
MENUITEM=FFORMAT5,Format DiskOnChip2000 with TrueFFS v5.0
MENUITEM=FFORMAT4,Format DiskOnChip2000 with TrueFFS v4.2
MENUITEM=MSDOS,Exit to MSDOS
MENUDEFAULT=LLOCAL,5

[LLOCAL]
[FFORMAT5]
[FFORMAT4]
[MSDOS]
[COMMON]
```

AUTOEXEC.BAT

```
. . .
SET PATH=c:\;C:\DOS;c:\DOC-V42
GOTO %CONFIG%
:FFORMAT5
  echo "formatting DiskOnChip2000 using doc504.exb"
  cd \DOC-V50
  dformat /win:d600 /s:doc504.exb /y
  goto end
:FFORMAT4
  echo "formatting DiskOnChip2000 using doc42.exb"
  cd \DOC-V42
  dformat /win:d600 /s:doc42.exb /y
  goto end
:LLOCAL
  \LINUX\loadlin \linux\lin247c\bzimage root=/dev/hda1  ro
  GOTO END
:MSDOS
:END
```

3.0 Linux Development Environment

The development system used for the embedded system is based on the standard REDHAT v7.2 distribution. LINUX may be installed from CD to the hard disk. The disk should already have been partitioned something like:

```
Disk /dev/hda: 255 heads, 63 sectors, 524 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1             5       495   3911827+  83  Linux
/dev/hda2   *         1         4     32098+   4  FAT16 <32M
/dev/hda3           496       524    265072+  82  Linux swap
```

LINUX should be installed onto device `/dev/hda1` with `/dev/hda3` used for swap. The LINUX disk partitions will need to be initialised appropriately. Take care not to initialise the DOS partition (`/dev/hda2`).

The installation procedure should automatically detect any hardware devices present such as keyboard, mouse and video. It should also detect the Ethernet interface. Setup and appropriate IP address for your network. At this stage it is advisable not setup any firewall configuration.

No boot loader should be installed for LINUX as we are using LOADLIN as described above, however a boot floppy disk should be created. This disk will be needed to boot LINUX until a suitable kernel has been built and copied to the DOS partition for LOADLIN.

A minimalist custom installation should be selected with general development and network options but with no X-windows or window managers installed. Ensure that you provide a password for the root user but it is not necessary at this stage to setup any other user accounts.

3.1 Custom Kernel Configuration

Having successfully loaded REDHAT 7.2 you should be able to boot LINUX using the boot disk created above. Login to the root account using the password defined.

The standard set of kernel sources supplied with REDHAT 7.2 are based on kernel release 2.4.7. In order to develop the system for PROTOR a custom kernel needs to be built with additional support for Memory Technology Devices (MTD) in order to support the DiskOnChip device.

Kernel 2.4.7 contains general drivers and development for solid-state flash disks such as the M-System DiskOnChip.

Note it is possible to use binary drivers available from M-Systems however, being binary, these drivers contravene the Open Source GPL licence and should not be released as part of a production item. It was therefore decided to use the general MTD drivers available within the Linux kernel sources.

The latest set of kernel sources for MTD should be downloaded from the web site www.linux-mtd.infradead.org and loaded on top of the standard 2.4.7 sources. This set of sources also contains some useful utilities for MTD development.

The kernel sources are contained in directory `/usr/src/linux-2.4.7-10`. To build a custom kernel use the command.

```
make menuconfig
```

from within this directory.

The following is a brief summary of the important kernel options to be selected within the MTD section. This shows only the main options turned on or included within the kernel. Most other options may be selected as modules and loaded at run-time in order to keep the resultant kernel size to a minimum.

Section	***Option***	***Reason***
• Memory Technology Devices (MTD)	CONFIG_MTD=y	Include MTD support
	CONFIG_MTD_DEBUG=y	Turn on Debug
	CONFIG_MTD_DEBUG_VERBOSE=0	Verbose level = 0 (quietest)
	CONFIG_MTD_CHAR=y	
	CONFIG_MTD_BLOCK=y	
	CONFIG_FTL=y	
	CONFIG_NFTL=y	
	CONFIG_NFTL_RW=y	Turn on Read/Write to MTD device
• Self	CONFIG_MTD_DOC2000=y	DiskOnChip 2000

	Contained MTD devices	CONFIG_MTD_DOCPROBE=y	Probe for address
		CONFIG_MTD_DOCPROBE_ADVANCED=y	Advanced Probe option
		CONFIG_MTD_DOCPROBE_ADDRESS=D6000	DOC2000 as specific address D6000
•	NAND Flash Device Drivers	CONFIG_MTD_NAND=y	Include NAND support
		CONFIG_MTD_NAND_ECC=y	Include software ECC

With this configuration we are enabling support for the DiskOnChip 2000 specifically and also at the address `0Xd6000` which is the default for the DiskOnChip on the Nova range of processor cards.

3.2 Building a Custom Kernel

Build the new kernel with the command

```
make dep clean bzImage modules modules_install
```

The resultant kernel and its equivalent system map are found in the files

```
/usr/src/linux-2.4.7-10/arch/i386/boot/bzImage  and
/usr/src/linux-2.4.7-10/System.map
```

3.3 Booting Custom Kernel

To boot the system from this new kernel you will need to mount the DOS partition and copy both these files to the appropriate directory. For example

```
mkdir /mnt/dos
mount /dev/hda2 /mnt/dos -t msdos
```

In the example AUTOEXEC.BAT file shown above the directory used for storage of the Linux kernel is C:\LINUX\LIN247C. Therefore the new kernel image may be copied using the command

```
cp /usr/src/linux-2.4.7-10/arch/i386/boot/bzImage
          /mnt/dos/linux/lin247c/bzimage
```

Before booting the new kernel it is necessary to create device entries within the system for the DiskOnChip. A script exists within in the MTD utilities which may be downloaded from the MTD web site as described above. If this software is downloaded and unpacked into directory `/usr/protor4/mtd` then action this script by the commands:

```
cd /usr/protor4/mtd/util
./MAKEDEV
```

The devices created should be as follows and have a major node of 93.

```
ls -l /dev/nft*
brw-r--r--    1 root     root      93,   0 Sep  3 15:23 nftla
brw-r--r--    1 root     root      93,   1 Sep  3 15:23 nftla1
brw-r--r--    1 root     root      93,   2 Sep  3 15:23 nftla2
```

You should now be able to shutdown Linux and reboot. The system boots to MS-DOS and shows the options defined in the CONFIG.SYS file. Firstly select the option

```
Format DiskOnChip using TrueFFS v4.2
```

The DiskOnChip will be formatted , loosing any previous information stored on the device. This procedure also loads software to the device which enables it to be detected by the system BIOS. Reboot the system again in order to detect the device.
Now select the option :

```
Boot Linux 2.4.7  kernel(MTD) root on /dev/hda1
```

The LINUX system should now boot. During the boot procedure there should be messages showing that the DiskOnChip has been detected. You should also see messages about the device `/dev/nftla`.

For example, following a reboot, try the command

```
dmesg | grep DiskOnChip
```

Which should show messages similar to the following if the device has been correctly detected.

```
Using configured DiskOnChip probe address 0xd6000
DiskOnChip 2000 found at address 0xD6000
```

```
2 flash chips found. Total DiskOnChip size: 32 MiB
mtd: Giving out device 0 to DiskOnChip 2000
```

Additionally

```
dmesg | grep nftl
```

Should show something like:

```
NFTL driver: nftlcore.c $Revision: 1.86 $, nftlmount.c
$Revision: 1.28 $
 nftla:
```

3.4 Formatting DiskOnChip for Linux

Provided that the DiskOnChip has been detected then the chip can be formatted and mounted for use within LINUX. Before formatting the chip it is easiest to remove any existing partition information using the command

```
dd if=/dev/zero of=/dev/nftla count=1 bs=512
```

To test the device it should now be possible to format the device and mount it. For this application it is probably best to make a filesystem over the whole device rather than setup individual partitions. The following commands make a file system on the device, create a mount point and mount the device.

```
mke2fs /dev/nftla
mkdir /flash
mount /dev/nftla /flash –t ext2
```

You should now be able to copy files and read them from the DiskOnChip device using the mount point `/flash`.

3.5 Embedded system Utilities

Having now successfully installed and accessed the DiskOnChip it is now the aim to identify and load sufficient system files and application tasks onto the DiskOnChip to allow the system to be bootable and usable from stand-alone DiskOnChip , that is with no hard-disk present. The main challenge being the space available on the DiskOnChip. Searches on the Internet yielded a number of useful utilities designed for small-footprint , embedded applications. The applications chosen are described below.

a. BusyBox (www.busybox.net)

This single program can be used to emulate a large number of UNIX utilities. A configuration file used during the build process allows the utilities supported to be defined and hence control the size of the eventual task. The size of the single task is much less than the combined size of the equivalent Unix utilities. Another advantage is that the task can also built without `glibc NSS` support.

For the most recent C compilor (GNU C Lib 2.0) access to various system files and databases is controlled through the NSS suite. This requires the final system to have a large number of run-time libraries (`/lib/security/libnss*`). It would seem to be advantageous to build BusyBox without NSS support however for our application subsequent utilities such as FTP and TELNET require NSS support and so Busybox was built with NSS support included.

Installation procedure:

- Download latest source tree (busybox-0.60.3) from site
- Unpack into directory /usr/protor4/busybox-0.60.3
- Edit file busybox/Config.h to define entries to include
- Run makefile to build resultant task image
- Install the task onto DiskOnChip and make entry links to all included utilities.

b. TinyLogin (http://tinylogin.busybox.net)

From the same developers as BusyBox , this single module emulates a number of Unix processes for Login and access.

Installation procedure:

- Download latest source tree (tinylogin-1.0.2) from site
- Unpack into directory /usr/protor4/tinylogin-1.0.2
- Edit file busybox/Config.h to define entries to include
- Run makefile to build task image
- Install the task onto DiskOnChip and make entry links to all included utilities.

c. wu-ftpd (www.wu-ftpd.org)

In order to be able to copy files to and from the embedded system an FTP daemon is required. In our configuration the embedded system will be used in a private network and so we can afford to be more relaxed about security. REDHAT 7.2 make extensive use of the PAM security system and so the standard FTP daemon requires the additional overhead of the PAM run-time libraries. To avoid this requirement download the latest set of sources for the `wu-ftpd` daemon. It is possible to build this daemon without PAM support.

Installation procedure:

- Download latest source tree (wu-ftpd-2.6.2) from site.
- Unpack into directory /usr/protor4/wu-ftpd
- Configure software to disable PAM facilities and make new distribution :

```
./configure --disable_pam
make
```

- The resultant executable (`../bin/ftpd`) needs to be copied to the flash disk (`../sbin/in.ftpd`) along with the various access files (`/etc/ftp*`)
- This FTP daemon requires a number of runtime libraries to be present. These are:

```
/lib/libcrypt.so.1
/lib/libnsl.so.1
/lib/libresolv.so.2
```

d. TELNET server

Prosig have developed their own simple telnet style command server. This process attaches to a specific socket and waits for a connection from a telnet client. On connection a password command/response mechanism authenticates the user. Provided the correct password is given then system commands may be entered and responses echoed to the user.

Installation procedure:

- Download latest source tree from site.
- Unpack into directory /usr/protor4/command_server.
- Copy executable to flash disk (../usr/protor4)

3.6 Linux System for DiskOnChip

The tables provide in Appendix A describe the complete set of system files created for the DiskOnChip device in order to create a fully functional operational system. This set of files represented a small footprint system which allowed sufficient space on a 32Mbyte DiskOnChip device for the additional application code.

3.7 Making the DiskOnChip Bootable

In order to boot the embedded system from DiskOnChip then the kernel image and System map file created above need to be copied to the device into the directory `/boot`.

The image also needs to be modified to select the correct root file system when it boots. The kernel built on the development disk will expect the root file system on /dev/hda1. To change the kernel to boot with the root file system on the DiskOnChip use the command

```
rdev  /flash/boot/bzImage /dev/nftla
```

The standard boot loaders either LILO or GRUB cannot be used as they do not have in-built support for the DiskOnChip. However the MTD source tree downloaded above contains sources and an executable for a modified LILO which is DiskOnChip aware. This source tree also contains a boot block file suitable for use with the DiskOnChip. The following LILO configuration file (lilo.conf) was created and copied to the device.

```
boot = /dev/nftla
disk=/dev/nftla bios=0x80
image = /boot/bzImage
  root = /dev/nftla
  label = protor
  read-only
```

With the DiskOnChip mounted on /flash and the following files in the directory /flash /boot

```
BzImage
System.map
boot.b
```

Then issue the following command to create a boot block on the DiskOnChip

```
./lilo-mtd –r /flash –C /etc/ lilo.conf
```

The DiskOnChip should now be stand-alone and bootable. Shutdown the system. Remove the hard disk. Reboot and enter the system BIOS setup. Set the primary boot device for SCSI for the DiskOnChip. When the system now boots it should detect the DiskOnChip , find its boot block and load the kernel. When the kernel is loaded this should execute the `/sbin/init` program which when complete executes the script `/etc/init.d/rcS`.

3.8 Application Startup

As mentioned above, on boot the kernel is loaded from DiskOnChip and the file system mounted and started. The script `/etc/init.d/rcS` controls the startup and initialisation of individual files. The following is a suitable rcS script for a NOVA-7892 card and initialising PROTOR software.

```
#!/bin/bash
#
#  /etc/init.d/rcS  Single User Startup script for PROTOR
#
# 1.  Set system date/time from CMOS clock
  echo "Setting date…"
  /sbin/hwclock -s -u
# 2.  Mount /proc filesystem
  echo "Mounting proc…"
  /bin/mount -n -t proc /proc /proc
# 3.  Check flash filesystem every reboot
  echo "Checking filesystems…"
  /sbin/fsck.ext2 -a /dev/nftla1
# 4.  Remount flash filesystem read-write
  echo "Remounting flash filesystem as root (rw)…"
  /bin/mount -n -o remount,rw / > /etc/mtab
# 5.  Clear mtab and remove stale backups
  rm -f /etc/mtab
  rm -f /etc/mtab~ /etc/mtab~~
# 6. Enter root, /proc and (potentially)
#               /proc/bus/usb and devfs into mtab.
  mount -f /
```

```
    mount -f /proc
    [ -f /proc/bus/usb/devices ] && mount -f -t usbdevfs
  usbdevfs /proc/bus/usb
    [ -e /dev/.devfsd ] && mount -f -t devfs devfs /dev
  # 7. Setup path
    export PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/protor4
  # 8. Setup networking. Load driver and setup IP address
    echo "Setting up network interfaces…\n"
    modprobe eepro100
    /sbin/ifconfig eth0 `cat /usr/protor4/ipfile` up
    /sbin/route add default eth0
    echo ""
  # 9.  Initialise USB Controller and load devices
    echo "Initialising USB controller "
    modprobe usb-uhci
    mount -t usbdevfs usbdevfs /proc/bus/usb
  # 10. Startup system daemons ( syslogd, klogd, xinetd etc)
    chmod 777 /usr/protor4/*
    echo "Application startup…."
    echo "Starting network daemons …"
  #
  #  truncate the messages file
  #
    tail -n 100 /var/log/messages > /var/log/messages.0
    rm -f /var/log/messages
    echo "… syslogd " ; /sbin/syslogd -m 0
  #echo "… klogd " ; /sbin/klogd -2
    echo  "… xinetd " ; /usr/sbin/xinetd -stayalive
    echo  "… Protor command_server "
    /usr/protor4/command_server &
  # 11.  Startup PROTOR specific applications
    cd /usr/protor4
    startup
    /bin/bash
```

4.0 Summary

By following these procedures and loading the various software packages we have produced an embedded system which provides all of the necessary devices and drivers for a PROTOR system. The system supports the following key requirements of an embedded PROTOR system.

- System bootable from solid-state disk device (DiskOnChip). No requirement for floppy disk or hard disk devices.
- System supports USB devices.

- System supports Ethernet.
- Cut-down Linux installed to provide sufficient utilities for operation but remain within space constraints on DiskOnChip.
- Remote access utilities available for view and file transfer.

Appendix A

The following listing shows all files for the complete embedded small-footprint system for PROTOR. The majority of the files are taken from a standard Redhat 7.2 system together with additional files from the Busybox amd TinyLogin packages.

/ Directory

File	***Description***
/bin	Directory header
/boot	Directory header
/dev	Directory header
/etc	Directory header
/lib	Directory header
Linuxrc -> bin/busybox	Startup script for initrd (not used)
/lost+found	Directory header
/mnt	Directory header
/proc	Directory header
/sbin	Directory header
/usr	Directory header
/var	Directory header

/bin Directory

Most of the utilities in this directory are provided by the Busybox and Tinylogin modules. Each utility is simply a link to the appropriate module.

File	***Description***
addgroup -> tinylogin	
adduser -> tinylogin	
ash -> busybox	
bash	Bash shell (from Redhat 7.2 /bin/bash)
busybox	Busybox utility built from source.
cat -> busybox	
chgrp -> busybox	
chmod -> busybox	
Chown -> busybox	
Cp -> busybox	
Date -> busybox	
Dd -> busybox	
Delgroup -> tinylogin	
Deluser -> tinylogin	
Df -> busybox	
Dmesg -> busybox	
Echo -> busybox	
False -> busybox	
Grep -> busybox	
Gunzip -> busybox	
Gzip -> busybox	
Hostname -> busybox	
Kill -> busybox	

Ln -> busybox	
Login -> tinylogin	
Ls -> busybox	
Mkdir -> busybox	
Mknod -> busybox	
More -> busybox	
Mount -> busybox	
Mv -> busybox	
Pidof -> busybox	
Ping -> busybox	
Ps -> busybox	
Pwd -> busybox	
Rm -> busybox	
Rmdir -> busybox	
Sed -> busybox	
Sh -> busybox	
Sleep -> busybox	
Stty -> busybox	
Su -> tinylogin	
Sync -> busybox	
Tar -> busybox	
Tinylogin	Tinylogin module built from source
Touch -> busybox	
True -> busybox	
Umount -> busybox	
Uname -> busybox	
Vi -> busybox	
Zcat -> busybox	

/boot Directory

File	***Description***
Boot.5D00	Created by Lilo
Boot.b	Boot block file. From MTD distribution
BzImage	Kernel Image built for MTD support
Map	Map file created by Lilo
System.map	System map file built for MTD support

/dev Directory

File	***Major ID***	***Minor ID***	***Description***
Console	4	0	Console device
Fd0	2	0	Floppy Disk device
Hda	3	0	1st IDE Disk
Hda1	3	1	Disk partition #1
Hda2	3	2	Disk partition #2
Hda3	3	3	Disk partition #3
Hda4	3	4	Disk partition #4
Initrd	1	250	Initrd device
Kmem	1	2	Kmem device
Mem	1	1	Mem device
Nftla	93	0	MTD Flash Disk device
Nftla1	93	1	Flash Disk partiton #1
Nftla2	93	2	Flash Disk partition #2
Null	1	3	Null device

Ptyp0	2	0	
Ptyp1	2	1	
Ptyp2	2	2	
Ram	1	1	Ram device
Tty	5	0	Terminal Device
Ttyp0	3	0	Terminal #1
Ttyp1	3	1	Terminal #2
Ttyp2	3	2	Terminal #2
TtyS0	4	64	Com1 device
TtyS1	4	65	Com2 device
Zero	1	5	Zero device

/etc Directory

File	***Description***
Adjtime	
Fstab	File System table
Ftpaccess	FTP access file {
Ftpconversions	FTP access file {
Ftpgroups	FTP access file { from wu-ftpd distribution
Ftphosts	FTP access file {
Ftpusers	FTP access file {
Group	Group file
Hosts	Hosts file
hosts.conf	Hosts.conf file
init.d	Directory header
lilo.conf	Lilo configuration file
mtab	Mount table
nsswitch.conf	NSS configuration file
passwd	Password file
passwd-	Password backup
protocols	Protocols file
rc.d	Directory header
resolv.conf	Resolver configuration file
services	Services file
shadow	Shadow password file
shadow-	Shadow backup
syslog.conf	Syslog configuration file
xinetd.conf	Xinetd configuration file
xinetd.d	Directory header

/etc/init.d Directory

File	***Description***
rcS -> rcS.Nova7892	Link to appropriate startup script
rcS.Nova600	Startup script for Nova 600 card
rcS.Nova7892	Startup script for Nova 7892 card

/etc/rc.d Directory

File	***Description***
rc.sysinit	Main system startup script

/etc/xinetd.d Directory

File	***Description***
telnet	Xinetd configuration for telnet
wu-ftpd	Xinetd configuration for wu-ftpd

/lib Directory

This directory contains the runtime libraries required by the system utilities.

```
ld-2.2.4.so
ld-linux.so.2 -> ld-2.2.4.so
libc-2.2.4.so
libcom_err.so.2 -> libcom_err.so.2.0
libcom_err.so.2.0
libcrypt-2.2.4.so
libcrypt.so.1 -> libcrypt-2.2.4.so
libc.so.6 -> libc-2.2.4.so
libdl-2.2.4.so
libdl.so.2 -> libdl-2.2.4.so
libext2fs.so.2 -> libext2fs.so.2.4
libext2fs.so.2.4
libm-2.2.4.so
libm.so.6 -> libm-2.2.4.so
libnsl-2.2.4.so
libnsl.so.1 -> libnsl-2.2.4.so
libnss1_files-2.2.4.so
libnss1_files.so.1 -> libnss1_files-2.2.4.so
libnss_files-2.2.4.so
libnss_files.so.1 -> libnss1_files-2.2.4.so
libnss_files.so.2 -> libnss_files-2.2.4.so
libproc.so.2.0.7
libresolv-2.2.4.so
libresolv.so.2 -> libresolv-2.2.4.so
libtermcap.so.2 -> libtermcap.so.2.0.8
libtermcap.so.2.0.8
libutil-2.2.4.so
libutil.so.1 -> libutil-2.2.4.so
libuuid.so.1 -> libuuid.so.1.2
libuuid.so.1.2
```

/lib/modules Directory

File	***Description***
/lib/modules/2.4.7-10/modules.dep	Module dependencies
/lib/modules/2.4.7-10/modules.usbmap	USB map
/lib/modules/2.4.7-10/kernel/drivers/net/eepro100.o	Driver for Intel 82559 on Nova 7892
/lib/modules/2.4.7-10/kernel/drivers/net/eepro.o	"
/lib/modules/2.4.7-10/kernel/drivers/net/eexpress.o	"

/lib/modules/2.4.7-10/kernel/drivers/usb/uhci.o	USB drivers
/lib/modules/2.4.7-10/kernel/drivers/usb/usbcore.o	"
/lib/modules/2.4.7-10/kernel/drivers/usb/usbnet.o	"
/lib/modules/2.4.7-10/kernel/drivers/usb/usb-ohci.o	"
/lib/modules/2.4.7-10/kernel/drivers/usb/usb-uhci.o	"

/sbin Directory

Most of these utilities are linked to the Busybox module. The additional modules are highlighted.

File	***Description***
fsck.ext2	File system check for EXT2 (from Redhat 7.2)
getty -> ../bin/tinylogin	
halt -> ../bin/busybox	
hwclock	Hwclock read/write (Redhat 7.2)
ifconfig -> ../bin/busybox	
init -> ../bin/busybox	
insmod -> ../bin/busybox	
klogd -> ../bin/busybox	
lsmod -> ../bin/busybox	
mkfs.minix -> ../bin/busybox	
mkswap -> ../bin/busybox	
modprobe -> ../bin/busybox	
poweroff -> ../bin/busybox	
reboot -> ../bin/busybox	
rmmod -> ../bin/busybox	
route -> ../bin/busybox	
swapoff -> ../bin/busybox	
swapon -> ../bin/busybox	
syslogd -> ../bin/busybox	
update -> ../bin/busybox	

/usr/bin Directory

Most of these utilities are linked to the Busybox module. The additional modules are highlighted.

File	***Description***
basename -> ../../bin/busybox	
chvt -> ../../bin/busybox	
clear -> ../../bin/busybox	
cut -> ../../bin/busybox	
dirname -> ../../bin/busybox	
du -> ../../bin/busybox	
env -> ../../bin/busybox	
find -> ../../bin/busybox	
free -> ../../bin/busybox	
head -> ../../bin/busybox	
id -> ../../bin/busybox	
ipcrm	IPCRM utility from Redhat 7.2
ipcs	IPCS utility from Redhat 7.2
killall -> ../../bin/busybox	
logger -> ../../bin/busybox	

```
passwd -> ../../bin/tinylogin
reset -> ../../bin/busybox
sort -> ../../bin/busybox
tail -> ../../bin/busybox
telnet -> ../../bin/busybox
telnetd -> ../../bin/busybox
test -> ../../bin/busybox
tftp -> ../../bin/busybox
traceroute -> ../../bin/busybox
tty -> ../../bin/busybox
uniq -> ../../bin/busybox
uptime -> ../../bin/busybox
wc -> ../../bin/busybox
which -> ../../bin/busybox
whoami -> ../../bin/busybox
xargs -> ../../bin/busybox
yes -> ../../bin/busybox
```

/usr/sbin Directory

File	***Description***
chroot -> ../../bin/busybox	Link to Busybox
in.ftpd	FTP daemon built from wu-ftpd
in.telnetd	TELNET daemon
wu-ftpd -> in.ftpd	Link to FTP daemon
xinetd	Xinetd from Redhat 7.2

/var Directory

File	***Description***
log	Directory header for Log files
pid	Directory header for PIDs
run	Directory header for run PIDS