Linux Development Environment

User Guide

Issue 03

Date 2015-12-29

HiSilicon Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: http://www.hisilicon.com

Email: support@hisilicon.com

About This Document

Purpose

This document describes the high-definition (HD) Linux development environment, including setting up the Linux and network development environments, burning the HiBoot, Linux kernel, and root file system, and starting Linux-based applications.

This document helps you quickly understand the Linux development environment.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3798C	V1XX
Hi3796C	V1XX
Hi3798M	V1XX
Hi3796M	V1XX
Hi3798C	V2XX

Intended Audience

This document is intended for:

- Technical support personnel
- Software development engineers

Change History

Changes between document issues are cumulative. Therefore, the latest document issue contains all changes made in previous issues.

Issue 03 (2015-12-29)

This issue is the third official release, which incorporates the following change:

This document is updated to support the Hi3798C V200 solution, and chapter 2 is added.

Issue 02 (2015-05-06)

This issue is the second official release, which incorporates the following change:

Hi3798C V200 is supported.

Issue 01 (2014-10-30)

This issue is the first official release, which incorporates the following change:

Hi3796M V100 is supported.

Issue 00B01 (2014-06-13)

This issue is the first draft release.

Contents

Figures

Tables

1 Development Environment

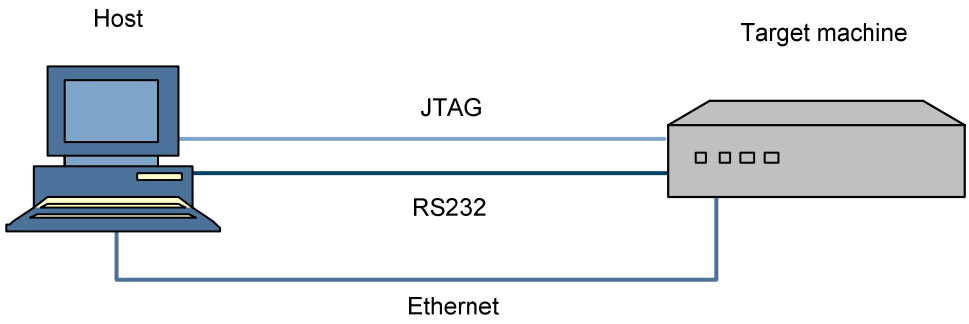
1.1 Embedded Development Environment

The embedded board is generally developed in cross compilation mode, that is, in host+target machine (evaluation board) mode. The host and target machine are typically connected over the serial port. They can also be connected over the network port or Joint Test Action Group (JTAG) interface, as shown in Figure 1-1.

The processors of the host and the target machine are different. A cross compilation environment suited for the target machine must be built on the host. After a program is compiled, connected, and located, an executable file is obtained. You can burn the executable file to the target machine and then run the program.

After the bootloader on the target machine is started, operation information of the target machine is transmitted to the host and displayed over the serial port or the network port. You can control the target machine by entering commands on the host's console.

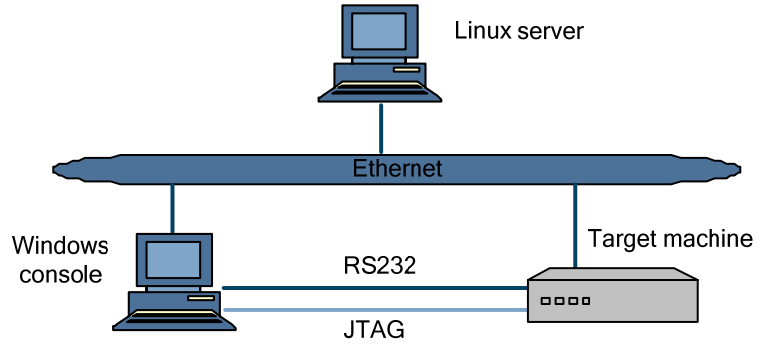
Figure 1-1 Embedded development environment



1.2 HD Linux Development Environment

The HD Linux development environment consists of a Linux server, a Windows console, and an HD reference board (target machine) on the same network, as shown in Figure 1-2.

Figure 1-2 HD Linux development environment



After a cross compilation environment is set up on the Linux server, and the Windows console connects to the HD reference board over the serial port or JTAG interface (the JTAG interface applies to software such as ADS/RealView Debugger), you can develop programs on the Windows console or on the Linux server through remote login. Table 1-1 describes the software running in the HD Linux development environment.

NOTE

Although a Windows console is used in the development environment, many operations performed by using the Windows console can also be completed on the Linux server by means of replacing the HyperTerminal with Minicom.

Table 1-1 Software running in the HD Linux development environment

Software		Description
Windows console	OS	Windows 98/Windows ME/Windows 2000/Windows XP/Windows 7
	Application software	PuTTY, HyperTerminal, TFTP server, and ADS/RealView Debugger (Windows 7 does not have inherited HyperTerminal and therefore serial port management software such as AbsoluteTelnet or SecureCRT is used.)
Linux server	OS	There is no special requirement on the release version, such as Redhat and Debian. The kernel version must be 2.6.9 or later. Ubuntu 10 or later is recommended. In addition, full installation is recommended.
	Application software	Network file system (NFS), telnetd, Samba, VIM, and ARM cross compilation environment (Gcc 4.4). Other application software varies according to the actual development requirements. Typically the required software is pre-installed by default. You need only configure the software before using it.
HD reference	Boot program	Fastboot 3.3.0/miniboot 1.0.0

Software		Description
board	OS	HiSilicon Linux (HiLinux for short). The HiLinux kernel is developed based on the standard Linux kernel V3.18.24, and the root file system is developed based on BusyBox 1.24.1.
	Application software	Common Linux commands such as **telnetd** and **gdb server**
	Program development library	The HD chip provides two tool chains: 32-bit Glibc 2.22 corresponding to compiler Gcc 4.9.2 and 64-bit Glibc 2.22 corresponding to compiler Gcc 5.1.
	GNU make	GNU Make 3.81 (If other versions are used, unpredictable compilation issues may occur.)

1.3 Setting Up the Linux Development Environment

1.3.1 Installing a Linux OS

You are advised to install a common Linux version for easy access to technical resources. For example:

- Later versions of RedHat, such as the RedHat Fedora Core series, Redhat Enterprise Linux, and Red Hat 3.4.4-2
- Earlier versions of RedHat, such as RedHat 9.0.

You are recommended to use the latest versions, such as the Fedora Core, SUSE series, and Ubuntu series, for the convenience of obtaining related resources.

The stable releases of Debian are also commonly used. If Debian is used, various software installation packages are available and can be easily updated online.

See chapter 7 "Setting Up the Linux Development Environment."

1.3.2 Installing the Cross Compiler

You are advised to use the tool chain in the SDK. For details about the installation, see **install_notes.txt** in the SDK.

CAUTION

A cross compiler obtained from other sources (such as Internet) may be incompatible with the existing kernel, and may result in unnecessary problems in the development process.

1.3.3 Installing the HD SDK

The HD SDK is a software development tool for the HD reference board. It contains all the tools and source code used in Linux-related application development. It is the basic software platform for HD chip development. For details about the installation procedures, see **install_notes.txt** in the SDK.

2 Boot Configuration

NOTE

This chapter describes the configuration method of the SDK to enable the boot to support multiple types of boards. Currently, only the Hi3798C V200 non-conditional access (CA) chip supports this function.

The boot supports multiple types of boards at the same time, and theses boards are distinguished through the ADC voltage. Currently, only the Hi3798C V200 platform supports this feature and supports at most six types of boards.

Table 2-1 shows the mapping between the default board types released in the SDK and the ADC voltages.

Table 2-1 Mapping between the default board types of the SDK and the ADC voltages

Board Type	Demo Board Name	ADC Voltage (V)
0	Hi3798CV2DMB	3.3
1	Hi3798CV2DMC	2.475
2	Hi3798CV2DMD	1.925
3	TBD	1.375
4	TBD	0.825
5	TBD	0

Configure the boot to support multiple board types in the SDK by running the following commands:

```
make menuconfig
    ->Board  --->
        ->Boot Regfile Config List  --->
(hi3798cv2dmb_xxx0.reg) Boot Reg File 0 (3.3V) //The ADC voltage is 3.3 V
and the board type is 0.
(hi3798cv2dmc_xxx1.reg) Boot Reg File 1 (2.475V) //The ADC voltage is
2.475 V and the board type is 1.
```

```
(hi3798cv2dmd_xxx2.reg) Boot Reg File 2 (1.925V) //The ADC voltage is
1.925 V and the board type is 2.
(hi3798cv2dme_xxx3.reg) Boot Reg File 3 (1.375V) //The ADC voltage is
1.375 V and the board type is 3.
(hi3798cv2dmf_xxx4.reg) Boot Reg File 4 (0.825V) //The ADC voltage is
0.825 V and the board type is 4.
(hi3798cv2dmg_xxx5.reg) Boot Reg File 5 (0V) //The ADC voltage is 0 and
the board type is 5.
```

CAUTION

The IDs of the board type are 0 to 5, which correspond to the ADC voltages and cannot be changed randomly.

At most six types of boards are supported.

If the number of supported board types is less than six, leave the corresponding table configuration option blank.

The advanced CA chip does not support this function.

3 Linux Kernel

3.1 Kernel Source Code

The kernel source code is stored in **source/kernel** of the SDK (as compressed package and patches). You can directly go to the directory and perform related operations. (Unless otherwise specified, the following sections take HiSTBLinux V100R005 as an example.)

3.2 Configuring the Kernel

The default kernel configuration for the corresponding HD chip in the SDK is recommended. You can run **make menuconfig** in the SDK and choose **Kernel** to select the required chip configuration.

You can also modify the default configuration as required.

CAUTION

The default kernel configuration in the SDK is optimized and adapted for the HD chips. If the default configuration is modified, some components may be unable to work. If you are not sure whether the modification is feasible, contact the HiSilicon FAE.

To modify the kernel configuration (for example, to modify **hi3798cv200_defconfig** by using arm-hisibv310-linux-gcc), do as follows:

```
Hisilicon#cd source/kernel/linux-3.18.y
Hisilicon#make ARCH=arm CROSS_COMPILE=arm-histbv310-linux-hi3798cv200_defconfig
Hisilicon# make ARCH=arm CROSS_COMPILE=arm-histbv310-linux- menuconfig
Hisilicon#cp .config arch/arm/configs/hi3798cv200_ defconfig
```

Then you can run **make menuconfig** in the SDK and choose **Kernel** to select the modified configuration.

NOTE

You can replace **make menuconfig** with **make config** or **make xconfig**, but the interface of **make config** is complex and **make xconfig** needs the support of XWindow. Therefore, the **make menuconfig** command is recommended, because its interface is intuitive and convenient for remote operations.

3.3 Compiling the Kernel

After saving the configuration, run the following commands in the SDK root directory to compile the kernel:

```
make linux;
make linux_install
```

After compilation, the generated kernel is stored in **pub/image/hi_kernel.bin** of the SDK.

To delete the processing files generated during compilation, run the following command:

```
make linux_clean;
```

To compile the kernel in the kernel source code directory, run the following commands:

```
cd source/kernel/linux-3.18.y
make ARCH=arm CROSS_COMPILE=arm-histbv310-linux- uImage
```

The generated image is named **uImage** in **linux-3.18.y/arch/arm/boot**.

NOTE

If an error occurs during compilation, run make ARCH=arm CROSS_COMPILE=arm-histbv310-linux- clean, followed by make ARCH=arm CROSS_COMPILE=arm-histbv310-linux- menuconfig, reload configuration files, and then run make ARCH=arm CROSS_COMPILE=arm-histbv310-linux- uImage.

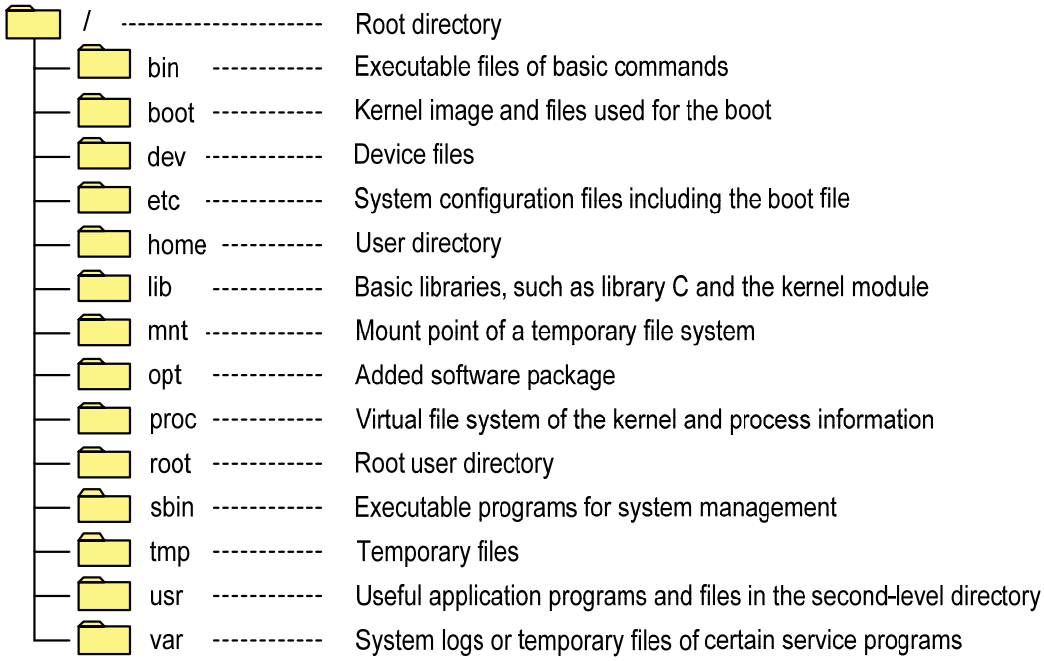
4 Root File System

4.1 Introduction to the Root File System

The top layer in the Linux directory structure is the root directory called "/". After loading the Linux kernel, the system mounts a device to the root directory. The file system of the device is called the root file system. All the system commands, system configuration, and mount points of other file systems are all located in the root file system.

The root file system is typically stored in the common memory, flash memory, or network-based file system. All the applications, libraries, and other services required by the embedded system are stored in the root file system. Figure 4-1 shows the top directory of the root file system.

Figure 4-1 Structure of the root file system top directory



A common Linux root file system consists of all the directories shown in the root file system's top directory structure. In an embedded system, the root file system needs to be simplified. Table 4-1 describes the directories that can be ignored in the embedded system.

Table 4-1 Directories that can be ignored in the embedded system

Directory	Description
/home, /mnt, /opt, and /root	Directories that can be expanded by multiple users
/var and /tmp	The **/var** directory stores system logs or temporary service program files. The **/tmp** directory stores temporary user files.
/boot	The **/boot** directory stores the kernel image. During startup, the PC loads the kernel from the **/boot** directory. However, for an embedded system, the kernel image is stored in the flash memory or on the network server instead of the root file system to conserve space. Therefore, this directory can be ignored.

NOTE

Empty directories do not increase the size of a file system. If there are no special reasons, it is recommended that you retain these directories.

4.2 Creating a Root File System by Using the BusyBox

Before creating a root file system by using the BusyBox, you need to obtain the BusyBox source code, and then configure, compile, and install the BusyBox.

4.2.1 Obtaining the BusyBox Source Code

After the SDK is installed, the BusyBox source code can be obtained in **source/rootfs/busybox/busybox-1.24.1**. You can also download the source code from http://www.busybox.net.

4.2.2 Configuring the BusyBox

Go to the directory where BusyBox is located and run the following commands (assume that the default BusyBox configuration file is ***xxx*.config**):

```
hisilicon$cd source/rootfs/busybox/
hisilicon$tar xf busybox-1.24.1.tar.bz2
hisilicon$cd busybox-1.24.1
hisilicon$ cp busybox-1.24.1.config/xxx.config .config
hisilicon$ make menuconfig
```

The configuration interface of the BusyBox is the same as that of the kernel. The function options are easy to understand. You can configure the BusyBox as required. Pay attention to the following two options that are displayed after you choose **BusyBox Settings** > **Build Options**:

```
[]Build BusyBox as a static binary (no shared libs)
(arm-XXXX-linux-) Cross Compiler prefix
```

The first option is used to determine whether to compile the BusyBox as an executable file with a static link. If the option is selected, the compiled BusyBox is a static link, which does not depend on the dynamic library and has a large size. If the option is deselected, the compiled BusyBox is a dynamic link, which has a small size but requires the support of the dynamic library. This option is deselected by default.

The second option is used to select a cross compiler and rename it **arm-XXXX-linux-** (the compiler name is subject to the name in the release version). After configuration, you need to save the settings and exit.

For details about the options of the BusyBox, see the *BusyBox Configuration Help*.

4.2.3 Compiling and Installing the BusyBox

To compile and install the BusyBox, run the following commands:

```
hisilicon$ make
hisilicon$ make install
```

If the BusyBox is successfully compiled and installed, the following directories and files are generated in the **_install** directory of the BusyBox:

```
drwxr-xr-x  2 linux  linux  4096 2005-04-22 11:01 bin
lrwxrwxrwx  1 linux  linux  11 2005-04-22 11:01 linuxrc->bin/busybox
drwxr-xr-x  2 linux  linux  4096 2005-04-22 11:01 sbin
drwxr-xr-x  4 linux  linux  4096 2005-04-22 11:01 usr
```

CAUTION

The BusyBox of HiSilicon support only GNU Make 3.81.

4.2.4 Creating a Root File System

After the SDK is installed, the created root file system is stored in the **pub/rootbox/** directory.

If necessary, you can create a root file system based on the BusyBox. The source code of the BusyBox is saved in **source/rootfs/busybox/**.

To create a root file system, perform the following steps:

Step 1 Run the following commands.

```
hisilicon$mkdir rootbox
hisilicon$cd rootbox
hisilicon$cp –R source/rootfs/busybox-1.24.1/_intsall/* .
hisilicon$mkdir etc dev lib lib64 tmp var mnt home proc
```

Step 2 Configured the required files in the **etc**, **lib**, and **dev** directories.

1. For the files in the **etc** directory, see the files in **/etc** of the system. The main files include **inittab**, **fstab**, and **init.d/rcS**. You are advised to copy these files from the **examples** directory of the BusyBox and then modify them as required.
2. You can copy the device files in the **dev** directory by running the **cp –R file** command or generate required device files by running the **mknod** command.
3. The **lib** directory is used to store the library files required by 32-bit applications. You need to copy related library files based on applications.
4. The **lib64** directory is used to store the library files required by 64-bit applications. Copy related library files according to applications. In addition, the soft link **ld-linux-aarch64.so.1 ->../lib64/ld-2.22.so** pointing to the 64-bit dynamic loader needs to be added in the **lib** directory.

----End

Upon completion of the preceding steps, a root file system is generated.

NOTE

Unless you have special requirements, the configured root file system in the SDK can be used directly. If you want to add applications developed by yourself, you only need to copy the applications and related library files to the corresponding directories of the root file system.

4.3 Introduction to File Systems

The common file systems in an embedded system include Squashfs, Cramfs, JFFS2, NFS, Yaffs2, UBIFS and EXT4. These file systems have the following features:

- Cramfs and JFFS2 have advantages in space and therefore are suited for embedded applications.
- Cramfs and Squashfs are read-only.
- JFFS2 is a readable/writable.
- NFS is suited for the debugging phase at the initial stage of development.
- Yaffs2 applies only to the NAND flash.
- UBIFS applies only to the NAND Flash. It is a log file system. Compared with Yaffs2, UBIFS has higher write efficiency.

NOTE

JFFS2, Cramfs, and Squashfs are used in the SPI flash, Yaffs2 and UBIFS apply only to the NAND flash, and EXT4 is typically used in eMMC.

4.3.1 Cramfs

Cramfs is a new file system designed for Linux kernel 2.4 and later. It is easy to use, easy to load, and has a high running speed.

The advantages and disadvantages of Cramfs are as follows:

- Advantages: Cramfs stores file data in compression mode. When Cramfs runs, the data is decompressed. This saves the storage space in the flash memory.
- Disadvantages: Cramfs cannot directly run on the flash memory because the stored files are compressed. When Cramfs runs, the data needs to be decompressed and then copied to the memory, which reduces the read efficiency. Also, Cramfs is read-only.

To enable the Linux running on the board to support Cramfs, you must add the **cramfs** option when compiling the kernel. After running **make menuconfig**, choose **File** > **systems**, select **miscellaneous filesystems**, and then select **Compressed ROM file system support**.

The mkfs.cramfs tool is used to create the Cramfs image. To be specific, the Cramfs image is generated after you process a created root file system by using mkfs.cramfs. This procedure is similar to that for creating an ISO file image using a CD-ROM. The related command is as follows:

```
hisilicon$mkfs.cramfs  ./rootbox ./cramfs-root.img
```

Where, **rootbox** is the created root file system, and **cramfs-root.img** is the generated Cramfs image.

4.3.2 Squashfs

Squashfs is also a compressed read-only file system. Compared with Cramfs, Squashfs supports greater compression ratio and larger images and files.

The kernel must be configured to support Squashfs. For details about the configuration, see section 3.2 "Configuring the Kernel."

```
File systems  --->
  [*] Miscellaneous filesystems  --->
<*>  SquashFS 4.0 - Squashed file system support  //Support Squashfs
 [*]    Include support for XZ compressed file systems //This option must
be selected if the Squashfs is compressed by using the XZ algorithm.
```

mksquashfs is a tool used to create the Squashfs image. The related command is as follows:

```
hisilicon$mksquashfs ./rootbox rootbox.squashfs -no-fragments -noappend -
comp xz
```

Where, **rootbox** is the file system directory, and **rootbox.squashfs** is the name of the file system image to be created.

4.3.3 JFFS2

JFFS2 is the successor to the JFFS file system created by David Woodhouse of RedHat. It is the actual file system used in original flash chips of embedded mini-devices. As a readable/writable file system with structured logs, JFFS2 has the following advantages and disadvantages:

- Advantages: The stored files are compressed. The system is readable and writable.
- Disadvantages: When being mounted, the entire JFFS2 needs to be scanned. Therefore, when the JFFS2 partition is expanded, the mounting time also increases. Flash memory space may be wasted if JFFS2 is used. The main causes are excessive use of log files and reclamation of useless storage units of the system. The size of wasted space is equal to the size of several data segments. Another disadvantage is that the running speed of JFFS2 decreases significantly when the memory is full or nearly full due to trash collection.

To load JFFS2, perform the following steps:

Step 1 Scan the entire chip, check log nodes, and load all the log nodes to the buffer.

Step 2 Collate all the log nodes to collect valid nodes and generate a file directory.

Step 3 Search the file system for invalid nodes and then delete them.

Step 4 Collate the information in the memory and release the invalid nodes that are loaded to the buffer.

----End

System reliability is improved at the expense of system speed. The loading process is even slower for flash chips with large capacity.

To enable the kernel to support JFFS2, you must select the **JFFS2** option when compiling the kernel (the kernel of HiSilicon supports JFFS2 by default). After running **make menuconfig**, choose **File** > **systems**, select **miscellaneous filesystems**, and then select **Journalling FLASH File System v2 (JFFS2) support**.

To create a JFFS2 file system, run the following command:

```
hisilicon$mkfs.jffs2 –d ./rootbox -l –e 0x20000 -o  jffs2-root.img
```

You can download the mkfs.jffs2 tool from the Internet or obtain it from the SDK. **rootbox** is a created root file system. Table 4-2 describes the JFFS2 parameters.

Table 4-2 JFFS2 parameters

Parameter	Description
d	Specifies the root file system.
l	Indicates the little-endian mode.
e	Specifies the flash block size.
o	Specifies the output image file.

4.3.4 NFS

Before using Cramfs and JFFS2, you need to burn the image of the root file system to the flash memory. The system loads the image from the flash memory during booting. At the initial stage of system development or porting, applications need to be added or modified frequently. Each time an application is modified, the image needs to be burnt again. This wastes time and affects the lifecycle of the flash memory.

As a distributed file system, NFS is used for sharing files and printers. It allows you to share files by invoking and mounting remote file systems. The usage of these file systems is the same as that of a local file system. The NFS adopts the client-server model. In this model, the server provides the directories to be shared, and the client mounts the directories and accesses files in the directories over the network.

If initramfs is used as the root file system, and NFS is mounted after the system boots, no operation on the flash memory is required. You can modify the applications on the Linux server. This is suited for the debugging phase of the development.

The method of configuring the NFS root file system on the Linux server is as follows: Edit the **exports** configuration file in **/etc**, add directories and parameters, and then run the **/etc/init.d/nfs start** command to start the NFS service.

The preceding operations must be performed by a super user and the exported directory must be an absolute path. If the NFS service is started, you only need to restart the NFS service after configuring files, that is, run **/etc/init.d/ nfs restart**.

After configuring the NFS root file system on the Linux server, mount the NFS on the board. The procedures are as follows:

Step 1 Create the initramfs image and start the system.

The kernel must be configured to support initramfs. For details about the configuration, see section 3.2 "Configuring the Kernel."

```
General setup  --->
     [ ] Initial RAM filesystem and RAM disk (initramfs/initrd) support
```

Enter the absolute path for the file system directory in the preceding square brackets ([]).

Compile the kernel. For details, see section 3.3 "Compiling the Kernel." The kernel image contains the file system. Download the image to the DDR or burn it to the flash memory to boot the system. You do not need to burn extra file system image.

Step 2 Run the following command to mount the NFS after the system boots:

```
mount –t nfs –o nolock,tcp, intr,soft,timeo=10,retrans=3
xx:xx:xx:xx:/rootfs_dir /mnt
```

You can then implement the development in the **/mnt** directory.

The preceding **mount** options indicate that the NFS is mounted in TCP mode. You can press **Ctrl+C** to block the operation of accessing the **/mnt** directory. If the operation is blocked for more than 3 seconds, an error code is returned. These operations ensure that the shell is not suspended when the network condition is poor and operations such as **ls** are executed in the **/mnt** directory.

----End

4.3.5 Yaffs2

Yaffs2 is an embedded file system designed for the NAND flash. It uses the log structure and provides loss equalization and power-off protection mechanisms to reduce the impact on the consistency and integrity of the file system due to power failures.

The advantages and disadvantages of Yaffs2 are as follows:

- Advantages
  - Is designed for the NAND flash, providing optimized software structure and fast running speed.
  - Stores the file organization information by using the spare area of the hardware. Only the organization information is scanned when the system boots. In this way, the system boots fast.
  - Adopts the multi-policy trash recycle algorithm, which improves the efficiency and fairness of trash recycle for loss balance.
- Disadvantages

  The stored files are not compressed. Even when the contents are the same, a Yaffs2 image is greater than a JFFS2 image.

In the SDK, Yaffs2 is provided as a module. To generate the Yaffs2 module, you need to add the path of the dependent kernel code to **Makefile** in the Yaffs2 code, and then perform compilation.

The Yaffs2 image, similar to the Cramfs image, is generated by using tools. You can enter **mkfs.yaffs2** to display parameter descriptions as follows:

```
hisilicon $ mkfs.yaffs2
mkyaffs2image: image building tool for YAFFS2 built Dec 25 2009
usage: mkyaffs2image dir image_file [convert]
       dir     the directory tree to be converted
       image_file      the output file to hold the image
       pagesize        the nand flash's pagesize.value in [2048,4096,8192]
is allowed.
       ecc_type        the ecc_type you will use in hisilicon nand flash
driver: hinand.
                      0:no ecc,1:1bit ecc, 2:4bit ecc, 3:8bit ecc, 4:24bit
ecc for 1k, 5:24bit ecc for 512.
       'convert'       produce a big-endian image from a little-endian
machine
```

Example:

```
hisilicon$ mkfs.yaffs2 ./rootfs/ ./images/rootfs_2k_1bit.yaffs2 2048 1
```

Where, **rootbox** is a created root file system, and **2k_1bit.yaffs2** is a generated Yaffs2 image, the value **2048** indicates the page size, and the value **1** indicates the error correcting code (ECC) type.

5 Burning the Kernel and Root File System

The HD reference board contains the DDR and flash memory. The DDR address space starts from 0x00000000, and the DDR size varies according to boards. For details about DDR size, see related hardware manuals.

For details, see the *Hi379X V100 Demo User Guide*.

For details about how to burn the boot, kernel, and root file system images, see the *HiBurn quick start video V1.0.exe*.

6 Application Development

6.1 Compiling Codes

You can choose a code compilation tool as required. Typically the Source Insight is used on Windows, and Vim+ctags+cscope is used on Linux.

6.2 Compilation Options

6.2.1 Using the VFP

You can generate the hardware floating point instruction by using the following compilation option and complete the floating point calculation by using the floating point unit (FPU).

```
-mfloat-abi=softfp -mfpu=vfpv3-d16
```

Sample:

```
CFLAGS+=-march=armv7-a -mcpu=cortex-a9 -mfloat-abi=softfp -mfpu=vfpv3-d16
```

NOTE

The vector floating point (VFP) is recommended due to its excellent compatibility.

6.2.2 Using the NEON

You can generate the NEON instruction by using the following compilation option and complete the floating point calculation by using the NEON module.

```
-mfloat-abi=softfp -mfpu=neon
```

Sample:

```
CFLAGS+=-march=armv7-a -mcpu=cortex-a9 -mfloat-abi=softfp -mfpu=neon
```

NOTE

The NEON compilation option can be used only when the CPU has the NEON module and the kernel supports the NEON module. Otherwise, the system reports an instruction error.

- To check whether the kernel supports NEON, run the following command on the board:

  ```
  cat /proc/cpuinfo
  ```

If the displayed **Features** row contains **neon**, the kernel supports NEON.

```
Features        : swp half thumb fastmult vfp edsp neon vfpv3 tls
```

- To check whether NEON is used in the image, run the following command on the Linux server:

  ```
  arm-histbv310-linux-readelf -A a.out
  ```

  If **Tag_Advanced_SIMD_arch: NEONv1** is displayed, NEON is used.

  ```
  Tag_FP_arch: VFPv3
  Tag_Advanced_SIMD_arch: NEONv1
  Tag_ABI_PCS_wchar_t: 4
  ```

6.3 Running Applications

Before running compiled applications, you need to add them to the target machine as follows:

- Add the applications and required libraries (if any) to the directories of the root file system of the target machine. Store applications in the **/bin** directory, library files in the **/lib** directory, and configuration files in the **/etc** directory.
- Create a root file system containing new applications.

NOTE

Before running applications, you need to write and read the file system. Therefore, choose the JFFS2 file system, or use the combination of the Cramfs and JFFS2 file systems.

It is recommended that you use the NFS in the debugging phase so that you do not need to recreate the root file system and burn files. You need to set and start the NFS service (for details, see section 4.3.4 "NFS"), and then mount the NFS directory to the directory of JFFS2 by running the following command:

```
mount -t nfs -o nolock serverip:path /mnt
```

serverip indicates the IP address of the server where the NFS directory exists. **path** indicates the path of the NFS directory on the server. After mounting, you need only to copy applications to the NFS system directory before running them on the target machine.

To create the Cramfs or JFFS2 file system, you need to create the file system image (see section 4.3 "Introduction to File Systems"), burn the root file system to the specified address in the flash (0x34200000) (see section 5 "Burning the Kernel and Root File System"), and set the boot parameters. In this case, new applications can run properly after Linux starts.

NOTE

To enable new applications to run automatically when the system starts, add the paths of the applications to be started in the **/etc/init.d/rcS** file.

6.4 Debugging Applications Using gdbserver

In most cases, applications need to be debugged. On Linux, the common debugging program is gdb. Because the resources for the embedded boards are limited, gdb is not used directly on the target machine. The gdb+gdbserver mode is typically used. gdbserver runs on the target machine, whereas gdb runs on the host. The root file system contains gdbserver. To debug an application using gdbserver, perform the following steps:

Step 1 Start Linux and log in to the shell.

Before debugging an application using gdb, you must start gdbserver. To be specific, you need to go to the directory where the application to be debugged is stored. For example, if the application **hello** needs to be debugged, run the following command:

```
hisilicon$ gdbserver :2000 hello &
```

The preceding command indicates that a debugging process is started on port 2000 of the target board, and **hello** is the application to be debugged.

Step 2 Start gdb on the Linux server. Because the kernel of the target machine is ARM, start arm-*xxxx*-gdb (the name is subject to the name of the released version).

Step 3 Enter the following command and connect the PC to the target machine.

```
(gdb) target remote 192.168.0.5:2000  /* 192.168.0.5 is the IP address of
the board. */
```

CAUTION

The port number in the preceding command must be the same as the number of the port enabled on the target machine.

Step 4 If the connection is successful, the following message is displayed:

```
remote debugging using 10.70.153.100:2000
0x40000a70 in ?? ()
```

Step 5 Load symbol files by running either of the following commands:

```
(gdb) add-symbol-file hello 40000a70
(gdb) file hello
```

Step 6 Enter gdb commands (such as **lis**t, **run**, **next**, **step**, and **break)** to debug the application.

----End

7 Setting Up the Linux Development Environment

It is recommended that you use late Linux releases, such as RedHat 9.0, Fedora Core, Debian, or Mandrake. This section uses Fedora Core 2.0 as an example to illustrate how to set up the Linux development environment.

7.1 Linux Configuration Options

You are advised to purchase the commercial distributions of Linux for which a 60 to 90 days telephone technical support is provided instead of downloading one from the Internet or obtaining from other channels.

Install the Linux by referring to Table 7-1, which describes the configuration options you should pay attention to.

Table 7-1 Linux configuration options

Configuration Option	Recommendation	Purpose
Default language	English	Avoids garbled characters during remote login because the terminal may not support other languages.
Disk partition	Automatic	Reserves sufficient hard disk space for installing the SDK.
Firewall	Disabled	Ensures that system services run properly.
Installation type	Full installation	Prevents from potential installation failures due to lack of necessary components.

7.2 Configuring Required System Services

To configure required system services, perform the following steps:

Step 1 Start Linux after it is installed and log in as the user **root**.

Step 2 Configure the Samba service to exchange files with Windows.

1. Find options for configuring the Samba service on the menu bar of FC2.
2. In the displayed configuration window, create a Samba user.
3. Add a sharing folder.
4. Test whether the Samba service can be accessed by a Windows server.

Step 3 Run the **/etc/init.d/ssh start** command to enable the secure shell (SSH) service. For FC2, the SSH service is enabled by default. You can log in to the Linux server using PuTTY on a Windows server.

Step 4 Run the **/etc/init.d/nfs start** command to enable the NFS service, edit the **exports** file in **/etc/**, and add an **NFS** directory. In this case, you can use the **NFS** folder of the board for accessing the server or directly use the **NFS** folder of the server as the root directory of the board (common NFS mode in debugging).

----End

A basic Linux environment is set up successfully. Then you can install the SDK. For details on how to install the SDK, see section 1.3.3 "Installing the HD SDK."

7.3 Requirements on the Kernel Version of the Linux PC

In Table 7-2, the earliest kernel version and the latest kernel version are the tested versions of the kernel in which the UBI tool can run and be used for proper compilation. As the Linux kernel has multiple versions, some kernel versions may be incompatible with some tools. Therefore, you need to view the actual compilation results and running status. Table 7-2 is only for reference.

Table 7-2 Requirements on the kernel version for the UBI tool

Tool	Earliest Kernel Version	Latest Kernel Version
mkfs.ubifs (a tool for creating the UBI file system)	Linux 2.6.22	Linux 3.4

A Abbreviations and Acronyms

A

ADS	ARM development suite
ARM	advanced RISC machine

C

CRAMFS	compressed RAM file system

D

DMS	digital media solution

E

ECC	error correcting code
ETH	Ethernet
ELF	executable and linkable format

F

FC2	Fedora Core 2.0

G

GCC	GNU compiler collection
GDB	GNU debugger
GNU	GNU's not Unix
GUI	graphical user interface

H

HD	high-definition
HiLinux	HiSilicon Linux

I

IP	Internet Protocol
IPTV	Internet Protocol television

J

JFFS2	journalling flash file system version 2
JTAG	Joint Test Action Group

N

NFS	network file system

O

OS	operating system

P

PC	personal computer

S

SDRAM	synchronous dynamic random access memory
SDK	software development kit
SPI	synchronous peripheral interface
SSH	secure shell

U

UBIFS	unsorted block image file system

T

TFTP	Trivial File Transfer Protocol

V

VFP	vector floating point

Y

Yaffs2	yet another flash file system version 2