

Linux 移植应用开发指南

文档版本 V2.1

发布日期 2025-09-29

概述

本文主要介绍 Linux 内核和文件系统的移植和使用。

为了便于说明，本文定义了以下术语：

术语	说明
<platform>	芯片平台代号，代表某个系列的芯片，例如 xmfalcon、xmorca
<soc>	芯片名，代表某个具体的芯片，例如 xm7606v13、xm7206v11a
<toolchain>	代表工具链，例如 arm-gcc12.2.0-linux、arm-gcc12.2.0-linux-uclibceabi

关于<platform>：

<platform>	说明	对应的<soc>
xmfalcon	xm7606v1x 系列芯片平台代号	xm7606v13 xm7605v13 xm7605v12 xm7605v11 xm7506v13
xmorca	xm7206v1x 系列芯片平台代号	xm7206v11a xm7206v10b xm7206v10

关于<toolchain>：

<toolchain>	说明
arm-gcc12.2.0-linux	GCC12.2, ARM32 工具链, 配套 C 库为 Glibc-2.37, 仅适用于 xmfalcon 平台
arm-gcc12.2.0-linux-uclibceabi	GCC12.2, ARM32 工具链, 配套 C 库为 uClibc-ng-1.0.48, 仅适用于 xmorca 平台

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示如不可避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 警告	表示如不可避免则可能导致死亡或严重伤害的具有中等级风险的危害。
 注意	表示如不可避免则可能导致轻微或中度伤害的具有低等级风险的危害。

符号	说明
 须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2024-12-21	V1.0	DI 版本发布。
2025-03-04	V1.1	修正部分描述错误。
2025-05-28	V2.0	支持 xm7206v1x 系列芯片，考虑与 xm7606v1x 兼容，增加了一些术语定义便于说明。
2025-09-29	V2.1	增加 hpfat 使用说明。

目 录

前 言.....	i
1 Linux 内核.....	1
1.1 内核源代码	1
1.2 配置内核.....	2
1.2.1 在内核源码目录下配置:	2
1.2.2 在 SDK 环境配置.....	2
1.3 编译内核.....	3
1.3.1 在内核源码目录下编译.....	3
1.3.2 在 SDK 环境下编译	4
1.4 内核定制.....	4
1.4.1 网络支持.....	4
1.4.2 设备驱动支持	5
1.4.3 文件系统类型支持	6
1.4.4 内核镜像压缩	7
1.4.5 打印和调试信息.....	9
2 根文件系统.....	10
2.1 根文件系统简介	10
2.2 利用 busybox 制作根文件系统	11
2.2.1 获取 busybox 源代码.....	11
2.2.2 配置 busybox.....	11

2.2.3 编译和安装 busybox.....	12
2.2.4 制作根文件系统.....	13
2.3 文件系统简介.....	14
2.3.1 jffs2.....	14
2.3.2 yaffs2	15
2.3.3 ext4	16
2.3.4 squashfs.....	17
2.3.5 cramfs	18
2.4 文件系统定制.....	19
2.4.1 减小二进制文件大小.....	19
2.4.2 使用高压缩率的文件系统	20
3 附录.....	21
3.1 HPFAT 文件系统.....	21
3.1.1 HPFAT 文件系统简介.....	21
3.1.2 HPFAT 使用方法	21

表格目录

表 1-1 BSP 适配文件和目录的说明 1

表 2-1 根文件系统顶层目录结构说明 10

表 2-2 JFFS2 参数表 15

1 Linux 内核

1.1 内核源代码

内核源代码已存放于 SDK 目录下的 source/kerne/linux-5.10.y 目录中，其中包含 Linux Kernel 社区 (<https://www.kernel.org>) 原生的代码和基于原生代码对适配的 BSP 小系统驱动和模块。

表1-1 BSP 适配文件和目录的说明

目录名	描述
arch/arm/configs/<platform>_defconfig	<platform>内核默认配置文件，例如 xmfalcon_defconfig
arch/arm/configs/<platform>_quickstart_defconfig	快启的内核配置，一般命名中带有 quickstart
lotus/	bsp 小系统的驱动和模块代码，包含网口、Flash、SD 卡、USB、PWM、DVFS、DMA、时钟树、UART、I2C、SPI、RTC 等模块
lotus/machine/<platform>/dts/	<platform> dts (device tree) 文件
include/dt-bindings/clock/<platform>-clock.h	<platform>时钟树 ID 头文件
include/linux/lotus/	bsp 小系统的驱动和模块的头文件

1.2 配置内核

1.2.1 在内核源码目录下配置：

步骤 1 进入内核目录

```
cd source/kernel/linux-5.10.y/
```

步骤 2 生成.config 文件

```
make ARCH=arm CROSS_COMPILE=<toolchain>- <platform>_defconfig
```

或是直接拷贝：

```
cp arch/arm/configs/<platform>_defconfig .config
```

步骤 3 通过 “make menuconfig” 进行内核配置

```
make ARCH=arm CROSS_COMPILE=<toolchain>- menuconfig
```

步骤 4 根据实际需要，修改配置项，并保存退出（顶层界面上连续按 ESC 两次）。如果不了解 menuconfig 界面操作，可以参考界面顶部帮助提示。

步骤 5 回拷.config 文件

```
cp .config arch/arm/configs/<platform>_defconfig
```

步骤 6 清除源码目录中的临时文件：

```
make ARCH=arm CROSS_COMPILE=<toolchain>- distclean
```

----结束

1.2.2 在 SDK 环境配置

步骤 1 准备 SDK 配置。

拷贝需要的 SDK 配置到 SDK 根目录，详细的说明请参考《SDK 开发环境用户指南》。

步骤 2 SDK 下编译一次内核，详细的说明请参考《SDK 开发环境用户指南》。

```
make -j 20 linux
```

步骤 3 进入内核编译输出目录（通常在 SDK 的 out 目录下），运行 menuconfig。

```
cd out/<soc>/linux-5.10.y/  
make ARCH=arm CROSS_COMPILE=<toolchain>- menuconfig
```

步骤 4 根据实际需要，修改配置项，并保存退出。如果不了解 menuconfig 界面操作，可以参考顶部帮助提示。

步骤 5 拷贝修改后的配置到内核源码目录。

```
cp .config source/arch/arm/configs/<platform>_defconfig
```

----结束

1.3 编译内核

1.3.1 在内核源码目录下编译

步骤 1 进入内核源码目录

```
cd source/kernel/linux-5.10.y
```

步骤 2 配置编译环境

```
make ARCH=arm CROSS_COMPILE=<toolchain>- O=$(pwd)/../out <platform>_defconfig
```

步骤 3 编译内核

```
make ARCH=arm CROSS_COMPILE=<toolchain>- O=$(pwd)/../out ulmage -j 16
```

编译成功后，会在\$(pwd)/../out 目录下生成 arch/arm/boot/ulmage，将此文件烧录到单板上。

 说明

编译参数说明：

1. ARCH=arm 表示编译 arm 架构的代码；
2. CROSS_COMPILE 是指定工具链，注意<toolchain>后面要加横杆字符'-';
3. O=\$(pwd)/../out 指定编译输出文件的存放路径，\$(pwd)是获取当前目录，因此本示例中 O 的值为当前目录的上一级目录下的 out 目录，可以按需需改为其它目录，如果此目录不存在，内核会自行创建；
4. -j 16 是启动 16 个线程并行编译，可根据实际编译服务器的性能和内存资源调整。如果是在 PC 机本地建的虚拟机上编译，建减少并行编译线程数，否则可能造成编译过程中虚拟机因内存不足而卡死；
5. 编译成功会生成\$(pwd)/../out/arch/arm/boot/ulmage，此镜像可以用 bootm 命令引导。如果是快启场景，应该使用 arch/arm/boot/hwzImage-dtb，并用 boothz 命令引导。

步骤 4 如果需要清除编译产生的.o 和镜像，不包括.config 等配置文件

```
make ARCH=arm CROSS_COMPILE=<toolchain>- O=$(pwd)/../out clean
```

步骤 5 清除所有编译输出文件，包括配置文件

```
make ARCH=arm CROSS_COMPILE=<toolchain>- O=$(pwd)/../out distclean
```

或是

```
rm -rf $(pwd)/../out
```

----结束

1.3.2 在 SDK 环境下编译

编译内核:

```
make linux -j 16
```

清除内核编译输出:

```
make linux_clean -j 16
```

1.4 内核定制

1.4.1 网络支持

默认内核配置使能的有线网络和无线网络支持，以兼容大部分产品形态。

如果不需要无线网络，可以去掉无线网络模块的支持，需要完成以下步骤：

步骤 1 关闭对 IEEE802.11 协议公共类库的支持和对 Linux wireless LAN 配置 API 的支持。

```
[*] Networking support --->
[*]   Wireless ---->
< >   cfg80211 - wireless configuration API
```

步骤 2 关闭所有 IEEE802.11 协议相关的无线网络设备驱动。

```
Device Drivers --->
[*] Network device support --->
[ ] Wireless LAN ---->
```

如果不需要支持有线网口，则可以关闭以太网驱动和 PHY 的支持：

```
Device Drivers --->
```

```
[*] Network device support --->
[] Ethernet driver support --->
<> PHY Device support and infrastructure --->
```

同时关闭:

```
Lotus Support --->
Lotus Device Drivers --->
< > Ethernet select (Fast Ethernet MAC controller support) --->
< > Fast Ethernet MAC MDIO bus support
```

如果完全不需要支持网络, 关闭以下选项即可:

```
[*] Networking support --->
```

----结束

为方便开发调试, 默认内核配置使能了 NFS 文件系统, NFS 文件系统会消耗约 400K 内存, 如果最终量产版本不需要, 建议去掉, 关闭以下选项:

```
[] Network File Systems --->
```

关闭后, 如果需要再打开, 需要开启多个相关配置项:

```
[] Network File Systems --->
<*> NFS client support
<*> NFS client support for NFS version 2
<*> NFS client support for NFS version 3
[*] NFS client support for the NFSv3 ACL protocol extension
<*> NFS client support for NFS version 4
```

1.4.2 设备驱动支持

1.4.2.1 Flash 驱动

SDK 发布的默认内核配置同时支持 SPI-Nor 和 SPI-Nand, 而且, 同时开启了 Nand 常用的 Yaffs2 文件系统的支持和 Nor 常用的 jffs2 文件系统, 如果仅需要其一, 用户可以选择关闭另一种, 详细说明请参考 1.4.3 文件系统类型支持。

1.4.2.2 USB 相关的驱动

默认发布的配置同时支持的 USB Host 和 Device 功能, 如果只需要其一, 可以关闭另一种。

关闭 Host 功能:

```
Device Drivers --->
```

```
[*] USB support --->
< >   Support for Host-side USB
```

关闭 Device 功能:

```
Device Drivers --->
[*] USB support --->
< >   USB Gadget Support
```

如果完全不需要 USB 功能, 关闭以下选项即可:

```
Device Drivers --->
[*] USB support --->
```

如果仅需要关闭 U 盘支持, 可以关闭如下选项:

```
Device Drivers --->
[*] USB support --->
< >   USB Mass Storage support
```

1.4.3 文件系统类型支持

SDK 发布的内核配置默认关闭了不常用的文件系统配置, 如果有特殊需求, 用户需要在内核配置的 File systems 下使能相应的文件系统。

 说明

如果依赖 SDK 编译构建 rootfs, 需要打开 SDK 配置中相应的文件系统配置项:

```
Linux System --->
Filesystem --->
```

1. Ext4 文件系统的支持

内核配置通常默认支持 Ext4。如果不需要, 可以关闭以下选项。

```
File systems --->
<> The Extended 4 (ext4) filesystem
```

2. YAFFS2 文件系统

内核配置默认支持 Yaffs2, 如果不需要, 可以关闭如下选项:

```
Lotus Support --->
File systems --->
< >   yaffs2 file system support
```

3. UBIFS

内核配置默认不支持 UBIFS，如果需要 UBIFS，参考《UBI 文件系统使用指南》操作。

4. JFFS2 文件系统

内核配置默认支持 JFFS2，如不需要 JFFS2，可以关闭下面选项：

```
File systems --->
[*] Miscellaneous filesystems --->
<*> Journalling Flash File System v2 (JFFS2) support
```

5. SquashFS

SquashFS 是一种可用于 Flash 设备的只读文件系统，具有极高的压缩率，使用简单，性能优异，常用于存储介质非常有限的产品。

通常默认的内核配置关闭了对 SquashFS 的支持，如果需要使用 SquashFS，则应该打开下面选项：

```
File systems --->
[*] Miscellaneous filesystems --->
<*> SquashFS 4.0 - Squashed file system support
```

1.4.4 内核镜像压缩

linux 内核编译生成二进制文件 vmlinux 之后，需要通过某种压缩模式，将庞大的二进制文件压缩成体积更小的镜像文件，以便存放在空间有限的存储介质中；在系统启动时，会首先将压缩的镜像文件解压到 DDR 中，然后再从头执行，完成系统启动。对此，linux 内核提供了多种压缩方式：GZIP、LZMA、XZ、LZO 和 LZ4。而这几种压缩模式，在压缩率和解压速度之间各有千秋。

GZIP：linux 内核镜像默认的、也是最经典的压缩模式。它在压缩率和解压速度上，保持了最佳的平衡。

LZMA：LZMA 是 linux 内核新近才支持的压缩模式，相比另外两种压缩模式，它具有最高的压缩率（同样文件，通过 LZMA 压缩后的体积通常只有 GZIP 的 70%），但是压缩和解压缩的速度要差一些。适用于对启动时间要求不高且 SPI Flash 大小非常有限的场合中。

XZ：使用 LZMA2 压缩算法，生成的压缩文件比 POSIX 平台传统使用的 gzip、bzip2 生成的压缩文件更小，而且解压缩速度也很快。LZMA2 是 LZMA 的改进版本，相对于 LZMA，LZMA2 增进了 encoder 和 decoder 的实现，并改善了对多线程的支持。

LZO：这种压缩方式压缩率最低，但是压缩和解压的速度最快。

LZ4：与当前流行的压缩算法比较，LZ4 具有最快的压缩和解压速度，但是压缩比一般。

综合对比，在快启场景，应选择 GZIP 压缩并采用支持硬件 GZIP 解压的 boothz 命令引导。如果优先考虑节省 Flash 空间，可以选项 LZMA 压缩，以 2M 的内核估算，大小预估有~300K 的优化，但启动时间会增加~2s，实际不同版本可能有差异，以实际用户验证为准。

具体的选择方法（以下示例选择了 Gzip 模式）：

```
General setup --->
Kernel compression mode (Gzip) --->
(X) Gzip
() LZMA
() XZ
() LZO
() LZ4
```

如果选择 LZMA 压缩，boot 和内核都需要修改，具体修改步骤如下：

步骤 1 内核开启 LZMA 压缩

```
General setup --->
Kernel compression mode (Gzip) --->
() Gzip
(X) LZMA
() XZ
() LZO
() LZ4
```

步骤 2 如果内核没有开启 Appended DTB，需要开启

```
Boot options --->
[*] Use appended device tree blob to zImage (EXPERIMENTAL)
```

步骤 3 U-boot 开启 bootm 命令：

```
CONFIG_CMD_BOOTM=y
CONFIG_BOOTM_LINUX=y
```

步骤 4 boot env 中 bootcmd 用 bootm 引导内核：

```
bootcmd=sf probe 0;xmediaapp;sf read 0x41000000 0x100000 0x400000;bootm 0x41000000
```

须知

部分服务器可能不会默认带 LZMA 压缩工具，需要提前安装。

---结束

1.4.5 打印和调试信息

linux 内核中有不少和系统的调试信息相关内容，这部分内容在系统调试定位的时候非常重要，但也占用了一定的空间。在存储资源极度缺乏的环境中，也可以考虑将他们去掉。关闭它们不影响系统的正常运行。

内核跟踪器

```
Kernel hacking --->
[ ] Tracers --->
```

用户态出错信息

用户态程序出错后崩溃的时候，内核会打印一句简短的信息，告知出错的具体原因。该信息在应用程序调试阶段非常有用。关闭将不打印出错信息。

```
Kernel hacking --->
[ ] Verbose user fault messages
```


2 根文件系统

2.1 根文件系统简介

Linux 的目录结构的最顶层是一个被称为 “/” 的根目录。系统加载 Linux 内核之后，就会挂载一个设备到根目录上。存在于这个设备中的文件系统被称为根文件系统。所有的系统命令、系统配置以及其他文件系统的挂载点都位于这个根文件系统中。

根文件系统通常存放于内存和 Flash 中，或是基于网络的文件系统。根文件系统中存放了嵌入式系统使用的所有应用程序、库以及其他需要用到的服务。表 2-1 列出了根文件系统的顶层目录。

表2-1 根文件系统顶层目录结构说明

目录名称	描述
/	根目录
/bin	基本命令的可执行文件
/sbin	系统管理命令的可执行文件
/dev	设备文件
/etc	系统配置文件，包括启动脚本
/home	非 root 用户工作目录
/root	root 用户工作目录
/lib	基础库和驱动模块，例如 C 库、USB 驱动
/mnt	临时文件系统的挂载点

目录名称	描述
/proc	内核、进程信息的伪文件系统
/sys	设备和系统信息，内核设备模型与用户态的接口
/tmp	临时文件目录
/usr	用户的执行程序、配置信息等
/var	常用于存放系统日志、一些服务程序的临时文件

2.2 利用 busybox 制作根文件系统

利用 busybox 制作根文件系统需要先获取 busybox 源代码，然后配置、编译和安装 busybox，操作成功后开始制作根文件系统。

2.2.1 获取 busybox 源代码

成功安装 SDK 后，busybox 开源源代码就存放在 open_source/busybox 目录中。要获取最新 busybox 源代码也可以从网站 <http://www.busybox.net> 下载。

2.2.2 配置 busybox

修改 busybox 配置的方法：

步骤 1 编译一次 rootfs。

```
make -j 16 rootfs
```

步骤 2 进入 busybox 编译输出目录，运行 menuconfig。

```
out/<soc>/rootfs_buildir/busybox-1.36.1/  
make menuconfig
```

步骤 3 根据实际需要，修改配置项，并保存退出。如果不了解 menuconfig 界面操作，可以参考顶部帮助提示。

busybox menuconfig 配置菜单说明：

```
Busybox Settings ---> /* 关于busybox 的基础配置，下面会详细介绍 */  
--- Applets  
Archival Utilities ---> /* 与压缩解压文件相关的功能，请根据具体需要选择 */
```

```

Coreutils ---> /* busybox 核心命令集，请根据实际需求选择 */
Console Utilities ---> /* 控制台相关的命令 */
Debian Utilities ---> /* Debian 系统相关的功能，基本可以不选 */
Editors ---> /* 编辑器相关的功能，请根据需要选择 */
Finding Utilities ---> /* 与查找相关的功能，请根据需要选择 */
Init Utilities ---> /* 与系统启动相关的配置，必须保留 */
Login/Password Management Utilities ---> /* 登陆和用户密码的管理 */
Linux Ext2 FS Progs ---> /* 与 Ext2 文件系统相关的命令 */
Linux Module Utilities ---> /* 模块加载和卸载的命令 */
Linux System Utilities ---> /* linux 系统中各大模块的支持 */
Miscellaneous Utilities ---> /* 未分类的功能支持 */
Networking Utilities ---> /* 网络相关的支持，请根据实际需求选择 */
Print Utilities ---> /* 打印机相关的支持，如无特殊需求，可放心关闭 */
Mail Utilities ---> /* 和电子邮件相关的支持，如无特殊需求，可放心关闭 */
Process Utilities ---> /* 进程相关的功能，请根据实际需求，谨慎配置 */
Runit Utilities ---> /* 与系统服务相关的支持，可根据实际需求配置 */
Shells ---> /* 各种shell解释器配置 */
System Logging Utilities ---> /* 各种记录、日志相关的支持 */
---
Load an Alternate Configuration File
Save Configuration to an Alternate File

```

步骤 4 拷贝修改后的配置到 busybox 源码目录。

```
cp .config ../../../../source/rootfs/busybox/busybox-1.36.1.config
```

----结束

2.2.3 编译和安装 busybox

在 sdk 目录下，执行如下命令，可以编译和安装 rootfs 下的所有工具，包括 busybox。

```
make -j 16 rootfs
```

2.2.4 制作根文件系统

成功安装 SDK 后，可以整体编译 sdk 后，既可以生成客户选择的根文件系统镜像。

用户如有需要可在 busybox 的基础上制作根文件系统。

制作根文件系统的具体操作步骤如下：

步骤 1 mkdir rootfs

```
cd rootfs
```

```
cp -R out/<soc>/rootfs_builddir/busybox-1.36.1/.intsall/ .
```

```
mkdir etc dev lib tmp var mnt home proc
```

步骤 2 配置 etc、lib、dev 目录的必需文件。

- etc 目录可参考系统/etc 下的文件。其中最主要的文件包括 inittab、fstab、init.d/rcS 文件等，这些文件最好从 busybox 的 examples 目录下拷贝过来，根据需要自行修改。
- dev 目录下的设备文件，可以直接从系统中拷贝过来或者使用 mknod 命令生成需要的设备文件。拷贝文件时请使用 cp -R file。
- lib 目录是存放应用程序所需要的库文件，请根据应用程序需要拷贝相应的库文件。

----结束

完成以上两个步骤，一个完整的根文件系统就生成了。

说明

SDK 软件包中整体编译后已经包括配置好的完整的根文件系统，如果无特别需求，可直接使用。要添加自己开发的应用程序，只需将应用程序和相应的库文件拷贝到根文件系统的对应目录 out/<soc>/rootfs/，然后 make fs 一次即可。

须知

为了便于调试，默认发布的版本中没有设置 root 用户密码，如果有安全性设计考虑，请自行在最终产品中设置 root 密码。

2.3 文件系统简介

嵌入式系统中常用文件系统包括有 cramfs、jffs2、NFS、initrd、yaffs2、ext4 以及 squashfs、ubifs。它们的特点如下：

- cramfs 和 jffs2 具有好的空间特性，很适合嵌入式产品应用。
- cramfs 与 squashfs 为只读文件系统，目前只有 SPI Nor FLASH 支持这两种文件系统。
- squashfs 压缩率最高。
- jffs2 为可读写文件系统。
- NFS 文件系统适用于开发初期的调试阶段。
- yaffs2 文件系统只用于 NAND Flash。
- initrd 采用 cramfs 文件系统，为只读。
- ext4 文件系统用于 eMMC 卡。

2.3.1 jffs2

jffs2 是 RedHat 的 David Woodhouse 在 jffs 基础上改进的文件系统，是用于微型嵌入式设备的原始闪存芯片的实际文件系统。jffs2 文件系统是日志结构化的可读写的文件系统。

jffs2 的优缺点如下：

- 优点
使用了压缩的文件格式。最重要的特性是可读写操作。
- 缺点
jffs2 文件系统挂载时需要扫描整个 jffs2 文件系统，因此当 jffs2 文件系统分区增大时，挂载时间也会相应的变长。使用 jffs2 格式可能带来少量的 Flash 空间的浪费。这主要是由于日志文件的过度开销和用于回收系统的无用存储单元，浪费的空间大小大致是若干个数据段。jffs2 的另一缺点是当文件系统已满或接近满时，jffs2 运行速度会迅速降低。这是因为垃圾收集的问题。

加载 jffs2 文件系统时的步骤如下：

步骤 1 扫描整个芯片，对日志节点进行校验，并且将日志节点全部装入内存缓存。

步骤 2 对所有日志节点进行整理，抽取有效的节点并整理出文件目录信息。

步骤 3 找出文件系统中无效节点并且将它们删除。

步骤 4 最后整理内存中的信息，将加载到缓存中的无效节点释放。

----结束

由此可以看出虽然这样能有效地提高系统的可靠性，但是在一定程度上降低了系统的速度。尤其对于较大的闪存芯片，加载过程会更慢。

为了使内核支持 jffs2 文件系统，必须在编译内核时把 jffs2 的选项加入（我们发布的内核默认已经加入了支持）。在 make menuconfig 后，进入 “File systems”，选择 “Miscellaneous filesystems”，最后选中其中的 “Journalling Flash File System v2 (JFFS2) support” 选项（SDK 里面提供的内核默认已经选择了该文件系统的支持）。

jffs2 的制作方法为：

```
mkfs.jffs2 -d ./out/<soc>/rootfs -l -e 0x20000 -o jffs2-root.img
```

其中，mkfs.jffs2 工具可以从互联网中下载，也可以在 SDK 包中找到。./out/<soc>/rootfs 为之前已经制作好的根文件系统。参数说明如表 2-2 所示。

表2-2 JFFS2 参数表

参数	说明
d	指定根文件系统
l	little-endian 小端模式
e	Flash 的块大小
o	输出映像文件

2.3.2 yaffs2

yaffs2 是专门为 NAND Flash 设计的嵌入式文件系统。它是日志结构的文件系统，提供了损耗平衡和掉电保护，可以有效地避免意外掉电对文件系统一致性和完整性的影响。

yaffs2 的优缺点如下：

- 优点
 - 专门针对 NAND Flash，软件结构得到优化，速度快。
 - 使用硬件的 spare area 区域存储文件组织信息，启动时只需扫描组织信息，启动比较快。

- 采用多策略垃圾回收算法，能够提高垃圾回收的效率和公平性，达到损耗平衡的目的。
- 缺点
没有采用压缩的文件格式。当包含的内容相同时，yaffs2 镜像文件要比 jffs2 镜像文件大。

yaffs2 文件系统在 SDK 中作为一个模块提供。只需在 yaffs2 代码中的 Makefile 中加入所依赖的内核代码路径，进行编译，即可生成 yaffs2 文件系统模块。

yaffs2 镜像文件的制作和 cramfs 相同，即通过工具制作，只需简单的几个参数，具体如下：

对于 SPI Nand Flash：

```
mkyaffs2image ./out/<soc>/rootfs yaffs2-root.img [pagesize] [ecctype]
```

其中，./out/<soc>/rootfs 是之前已经制作好的根文件系统，yaffs2-root.img 是生成的 yaffs2 文件系统镜像文件，pagesize 是单板上焊接 NAND Flash 器件的页大小，ecctype 是单板上焊接 NAND Flash 器件的 ecc 类型。

2.3.3 ext4

ext4 是 linux 系统的第四代扩展文件系统，是 ext3 文件系统的后续版本，支持日志模式，可以防止掉电损坏文件系统。

Ext4 通常用于 eMMC 这种存储器件，不适用于 SPI Nor 和 Nand 器件，因为 ext4 有记录用户 ID，通常也不建议用在 U 盘这类外部存储设备。

步骤 1 制作支持 ext4 的内核镜像。

进入内核源码目录下，运行 menuconfig，使能以下内核配置后，重新编译内核：

```
File systems --->
<*> The Extended 4 (ext4) filesystem
[*] Use ext4 for ext2 file systems
```

步骤 2 制作 ext4 文件系统镜像。

在发布包 sdk/tools/utls/bin 目录下的 make_ext4fs 为制作 ext4 文件系统工具。使用方法如下：

```
./make_ext4fs -l 96M -s rootfs.ext4.img ./out/<soc>/rootfs
```

其中, -l 96M 是指定 ext4 的文件系统分区大小为 96MB, -s 为使用 gzip 压缩, rootfs.ext4.img 是生成的 Ext4 文件系统映像文件, ./out/<soc>/rootfs 是之前已经制作好的根文件系统目录。请根据实际情况修改参数。

----结束

2.3.4 squashfs

squashfs 文件系统是一套基于 Linux 内核使用的压缩只读文件系统, 压缩率高。

SquashFS 具有如下特点:

- 数据(data),节点(inode)和目录(directories)都被压缩
- 保存了全部的 32 位 UID/GIDS 和文件的创建时间
- 最大支持 4G 文件系统
- 支持检测并删除重复文件

SquashFS 被认为是 CramFS 的替代者, 相比 CramFS, 有以下优点:

1. 支持 GZIP、LZMA、LZO 等多种压缩方式, 可以有更好的压缩率;
2. 支持文件系统元数据的压缩
3. 支持不同块大小的压缩, 最大可以支持到 64KB 块, 而 CramFS 只能固定 4K 块大小
4. 能支持更大的文件系统
5. 有更完备的文件系统信息

以下是具体使用 Squashfs 的方法如下:

步骤 1 打开内核的 Squashfs 相关的配置选项。

```
File systems --->
[*] Miscellaneous filesystems --->
<*> SquashFS 4.0 - Squashed file system support
[*] Include support for XZ compressed file systems
```

步骤 2 制作 squashfs 镜像。

默认情况下 mksquashfs 工具制作的文件系统镜像采用 gzip 算法压缩, mksquashfs 还支持 xz 压缩算法。相同大小的文件系统, 采用 xz 压缩算法的镜像体积约为 gzip 压

缩算法的 4/5。要使 mksquashfs 支持 xz 压缩算法，在编译 mksquashfs 工具的时候，需要打开支持 xz 压缩算法的选项。

```
./mksquashfs rootfs ./ rootfs.squashfs -b 256K -comp xz
```

其中，rootfs 是之前已经制作好的根文件系统，rootfs.squashfs 是生成的 squashfs 文件系统映像文件。-b 256K 指定 squashfs 文件系统的块大小为 256K。-comp xz 制定文件系统压缩方式为 xz。

在 SDK 配置中支持编译出 SquashFS 格式的 rootfs 镜像，需要使能 squashfs rootfs 配置：

```
Linux System --->
Filesystem --->
[*] Create Squashfs Rootfs Image
```

步骤 3 修改 Boot env，bootargs 指定 rootfstype 为 squashfs。

假定原来为：

```
bootargs=mem=32M console=ttyAMA0,115200 root=/dev/mtdblock3 rootfstype=jffs2 rw
mtdparts=sfc:512K(boot),512K(bootargs),4M(kernel),11M(rootfs) lpj=9838592
```

改为：

```
bootargs=mem=32M console=ttyAMA0,115200 root=/dev/mtdblock3 rootfstype=squashfs
mtdparts=sfc:512K(boot),512K(bootargs),4M(kernel),11M(rootfs) lpj=9838592
```

注意，要去掉“rw”项。

----结束

2.3.5 cramfs

CramFS 是 Linux 内核 2.4 开始设计的一种只读文件系统，使用简单，是 SquashFS 的上一代只读文件系统，通常内核建议使用更优的 SquashFS 替代 CramFS。

如果因一些特殊原因必须要使用 CramFS，需要单独开启内核中的 CramFS 配置项，参考一下步骤：

步骤 1 进入内核源码目录，运行 menuconfig，打开一下选项后，重新编译内核

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

步骤 2 使用 mkfs.cramfs 来制作 cramfs 文件系统映像，具体操作如下所示：

```
mkfs.cramfs ./out/<soc>/rootfs ./rootfs.cramfs
```

其中，`./out/<soc>/rootfs` 是之前已经制作好的根文件系统目录，`rootfs.cramfs` 是生成的 `cramfs` 文件系统映像文件。

2.4 文件系统定制

默认 SDK 发布的 `rootfs` 增加较多功能，实际产品化可能需要裁剪，做小型化处理，文件系统的小型化可以从如下三个方面着手：

1. 通过配置 `busybox`，将不需要的功能、命令裁去，参考 1 2.2.2 配置 `busybox`；
2. 将文件系统上的可执行文件和库中多余的调试信息、符号信息删除掉，以减少文件系统的容量；
3. 采用更高压缩率的文件系统。

2.4.1 减小二进制文件大小

1. 去掉文件系统的可执行文件、相关的库文件中多余的调试信息和符号信息（即 `strip elf`）；
- 值得注意的是，文件系统的 `*.ko` 文件是不能 `strip` 的，否则 `ko` 文件不可用。

```
find rootfs/ -perm /u+x,g+x,o+x ! -name "*.ko" -exec <toolchain>-strip {} \;
```

上面命令将对 `rootfs` 目录下所有的可执行文件、库文件执行一次 `strip`。

SDK `menuconfig` 配置有配置开关可以 `strip` 默认的 `rootfs` 镜像：

```
Linux System ---->
Filesystem ---->
[*] Enable Strip
```

2. 剔除没有被使用的库文件；

文件系统的裁减，除了以上方式外，在所有开发工作都已经完成的阶段，还可以使用：找出文件系统中没有被使用的库文件，并删除。

下面的脚本可以帮我们找出哪些库文件从没被使用过：

```
#!/bin/bash
find rootfs/ -type f -perm /u+x,g+x,o+x -exec readelf -d {} \;>log1.txt
grep ".so" log1.txt | sort | uniq >log2.txt
grep "NEEDED" log2.txt >log3.txt
```

执行该脚本后，当前目录下会生成 `./log3.txt` 文件，在 `./log3.txt` 中列出的，就有被其它程序使用的库文件，这些库文件以及与之有链接关系的库文件都不能删除，其余的可删除。

2.4.2 使用高压缩率的文件系统

如果 rootfs 可以不需要写功能，可以考虑采用更高压缩率的只读文件系统，如 Squashfs。Squashfs 文件系统具有比 Jffs2 文件系统更高的压缩率，linux 内核支持 Squashfs 文件系统，但在默认配置可能没有开启 squashfs 选项。

具体使用 Squashfs 的方法参考 2.3.4 squashfs。

3.1 HPFAT 文件系统

3.1.1 HPFAT 文件系统简介

HPFAT 文件系统是 FAT32 文件系统的改进版本，在保留 FAT32 的基础上，新增了热插拔功能，显著提升了设备在动态插拔场景下的稳定性。

该文件系统实现热插拔功能的核心原理，是通过双 FAT 表（文件分配表）的备份机制确保关键数据不会出现异常。主 FAT 表用于日常文件分配与管理操作，备份 FAT 表则作为副本，在主表出现异常（如突发断电、热插拔过程中数据传输中断等情况）时，可快速启用以恢复文件系统结构，保障 FAT 表信息的正确性。同时文件系统采用写时复制的策略，先写文件数据再依次更新 FAT 表 and 文件目录项信息，确保在热插拔场景下，文件系统能维持可恢复的状态。

3.1.2 HPFAT 使用方法

在 `sdk/source/kernel/linux-5.10.y/arch/arm/configs/<platform>_defconfig` 文件中进行配置，添加或修改相关配置项：

配置项 `CONFIG_HPFAT_SUPPORT`

- 等于 `y` 时，开启 HPFAT 文件系统功能。
- 等于 `n` 时，关闭 HPFAT 文件系统功能。

配置项 `CONFIG_HPFAT_BUILD_TYPE`

- 等于 `y` 时，将 HPFAT 文件系统集成至内核，Linux 启动后可直接通过 `mount` 挂载到 FAT32 的存储设备上。

```
mount -t hpfat /dev/设备名 目标路径
```

- 等于 `m` 时，将 HPFAT 文件系统编译成 `hpfat.ko`，存放路径为 `sdk/out/<soc>/linux-5.10.y/lotus/fs/hpfat/hpfat.ko`。此模块没有依赖，可直接通过 `insmod` 加载，加载后即可通过 `mount` 挂载。