从零开始为 RISC-V 构建一个 Linux 系统

第 7 章 制作 Linux 内核

汪辰

目录

01

RISC-V 与 Linux

RISC-V Linux 的发展历史

时间点	关键事件与进展	主要意义
2010 年	美国加州大学伯克利分校推出 RISC-V 指令集架构	标志着 RISC-V 开放 ISA 的诞生
2015 年	RISC-V 基金会成立	将一个开放的技术指令集，转变为一个由全球产业共同维护、拥有法律保障、并能持续演进的开放计算生态。
2018 年 1 月 28 日	Linux 4.15 内核新增对 RISC-V 支持	标志着 RISC-V 首次获得 Linux 官方主流内核支持，是后续所有软件生态发展的基础。
2018 年 11 月	Linux 基金会与 RISC-V 基金会联合发布声明，宣布达成合作	两大开源组织的联合，为 RISC-V 生态提供了顶层的社区、资源和法律支持。
2020 年	RISC-V 基金会更名为 RISC-V 国际（RISC-V International），https://riscv.org/	保持其作为开放、中立技术标准组织的定位。同时与更多中国开源社区合作，如 openEuler、openKylin 等，共同推动 RISC-V 架构的生态适配。
2021 年 ~ 如今	2021 年 7 月，Ubuntu 发布首个面向 RISC-V 开发板（HiFive）的官方版本 2023 ~ 2024 年，Deepin 发布支持 RISC-V 的预览版系统；红帽 Fedora 对 RISC-V 的软件包编译完成度达 99%	更多主流和国产发行版加入，标志着 RISC-V 从内核支持扩展到完整桌面和服务器操作系统的成熟。

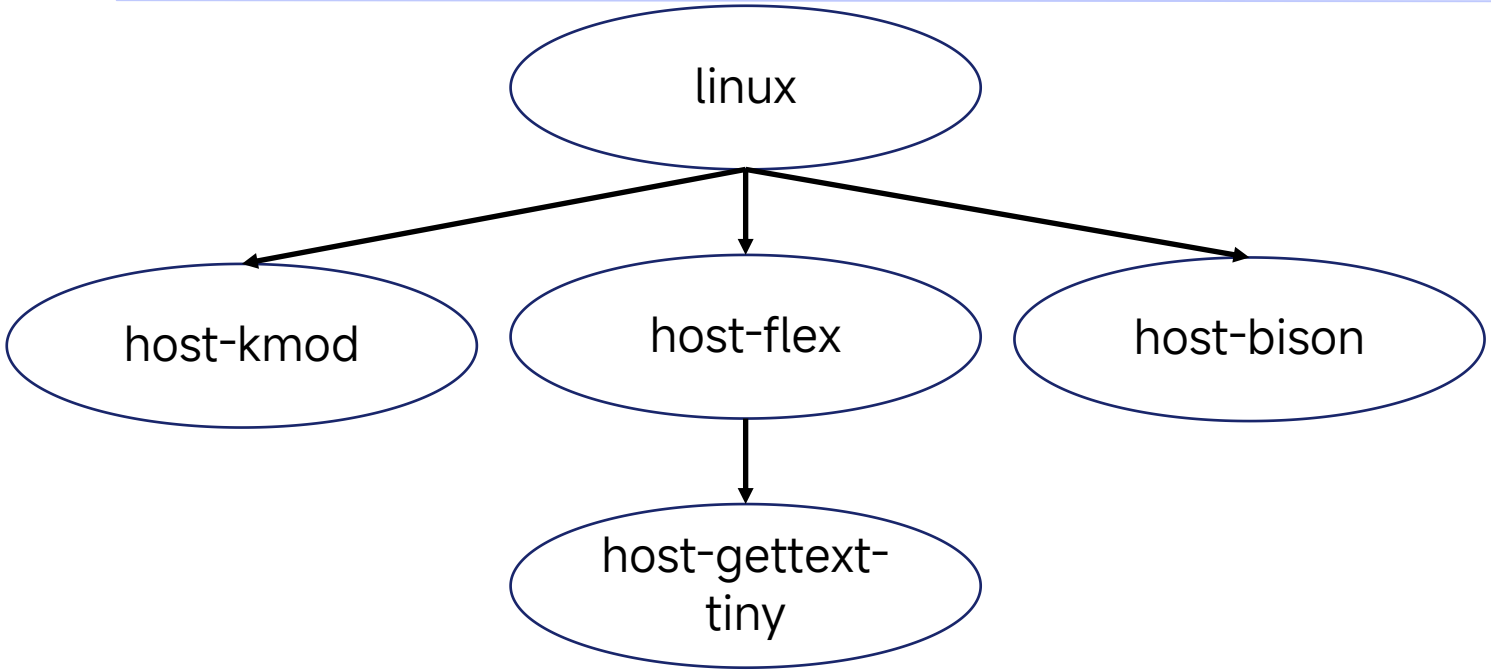
RISC-V Linux 的支持状态

支持领域	当前状态	详细描述
基础架构层面	稳定支持 riscv64	目前仅正式维护（Maintained）64 位版本（riscv64）和小端序
ISA 扩展	持续增加中	Linux 支持大量且不断增长的 RISC-V 指令集架构 (ISA) 扩展，包括标准扩展和厂商特定扩展，几乎每个内核版本都会新增扩展。Linux 对 RISC-V ISA 扩展的支持并非固定不变，而是持续增加。 为了应对 ISA 扩展的碎片化问题，RISC-V 引入了“Profiles”（如 RVA23），将标准扩展进行分组用于特定用例。
硬件和驱动	持续增加中	已广泛支持来自多个 Vendor 的多款 SoC（具体可以看内核源码树 arch/riscv/boot/dts），特别是最近新增了对服务器级处理器的支持，例如 64 核的 SG2042/SG2044。针对 GPU 和各种外围设备驱动程序支持也在积极推进。

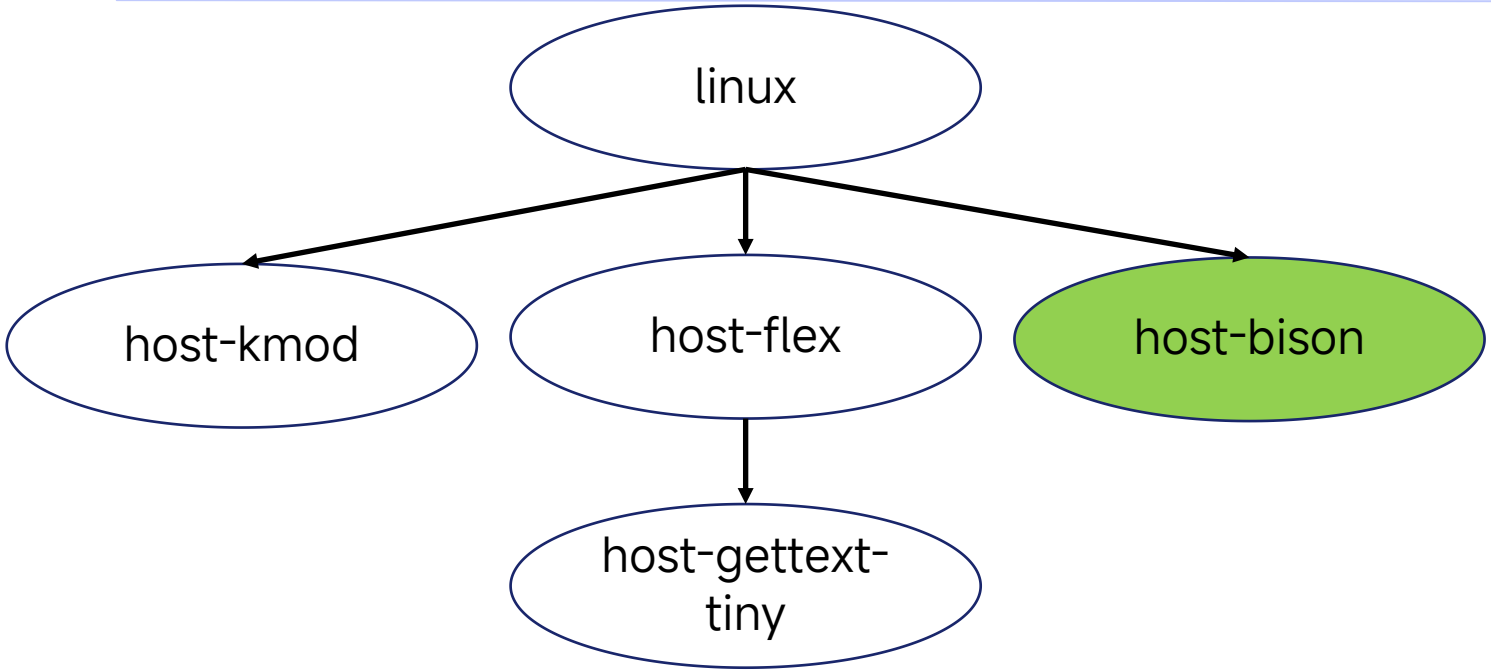
02

制作 Linux 内核

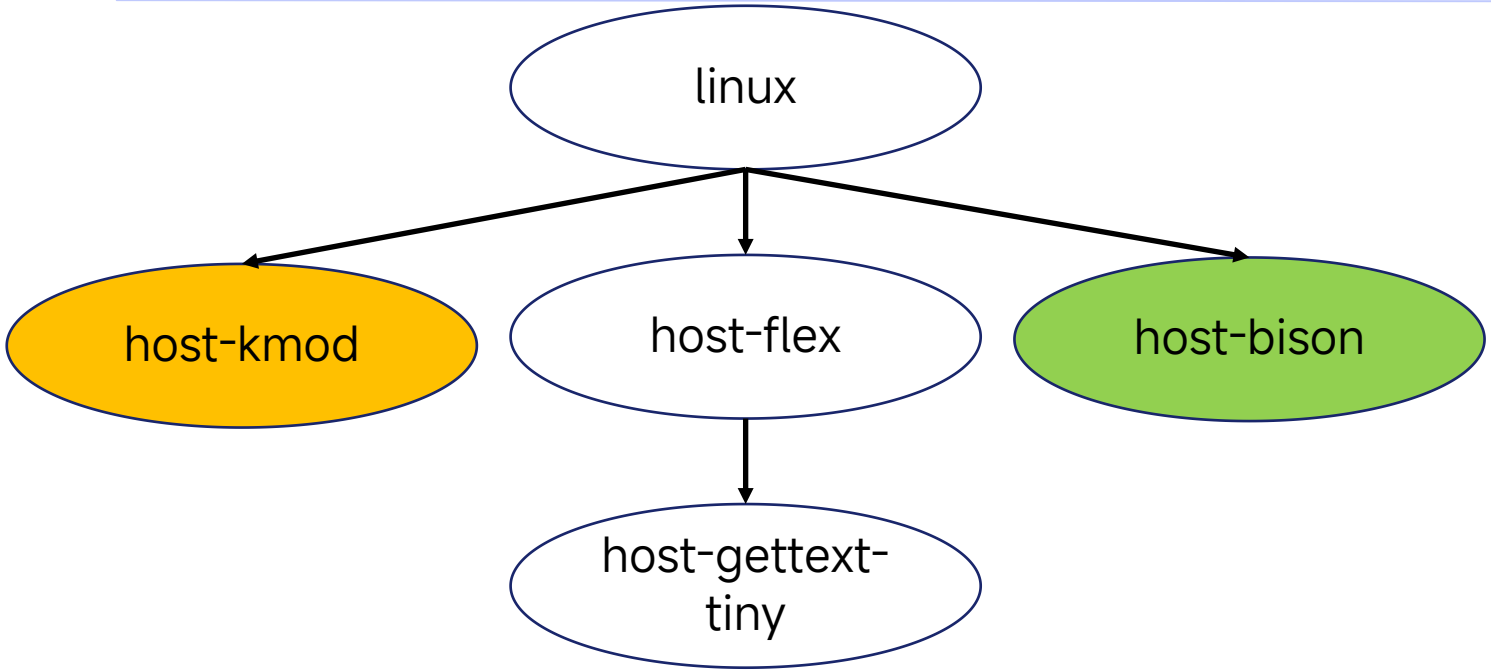
制作 Linux 内核



制作 Linux 内核



制作 Linux 内核



Kmod - 简介

Kmod 是 Linux 系统中一个基础且关键的工具集，主要用于管理和操作内核模块。工具集中包括了如下命令行工具（大部分命令都需要 root 权限执行）：

命令	用途
modprobe	智能加载模块，能自动解决依赖关系。首选此命令加载模块。
insmod	安装指定路径的模块文件，不处理依赖。
rmmod	从运行中的内核卸载模块。
lsmod	列出当前已加载的所有内核模块。
modinfo	显示某个模块的详细信息（如作者、参数等）。
depmod	分析模块间的依赖关系，生成依赖文件供 modprobe 使用。通常在安装新模块后运行。

Kmod – 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} CONFIG_SITE=/dev/null ./configure --
prefix=\"${HOST_DIR}\" --sysconfdir=\"${HOST_DIR}/etc\" --
localstatedir=\"${HOST_DIR}/var\" --enable-shared --disable-static --disable-gtk-doc --
disable-gtk-doc-html --disable-doc --disable-docs --disable-documentation --disable-
debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-tracking  --
disable-manpages --without-zlib --without-zstd --without-xz"
```

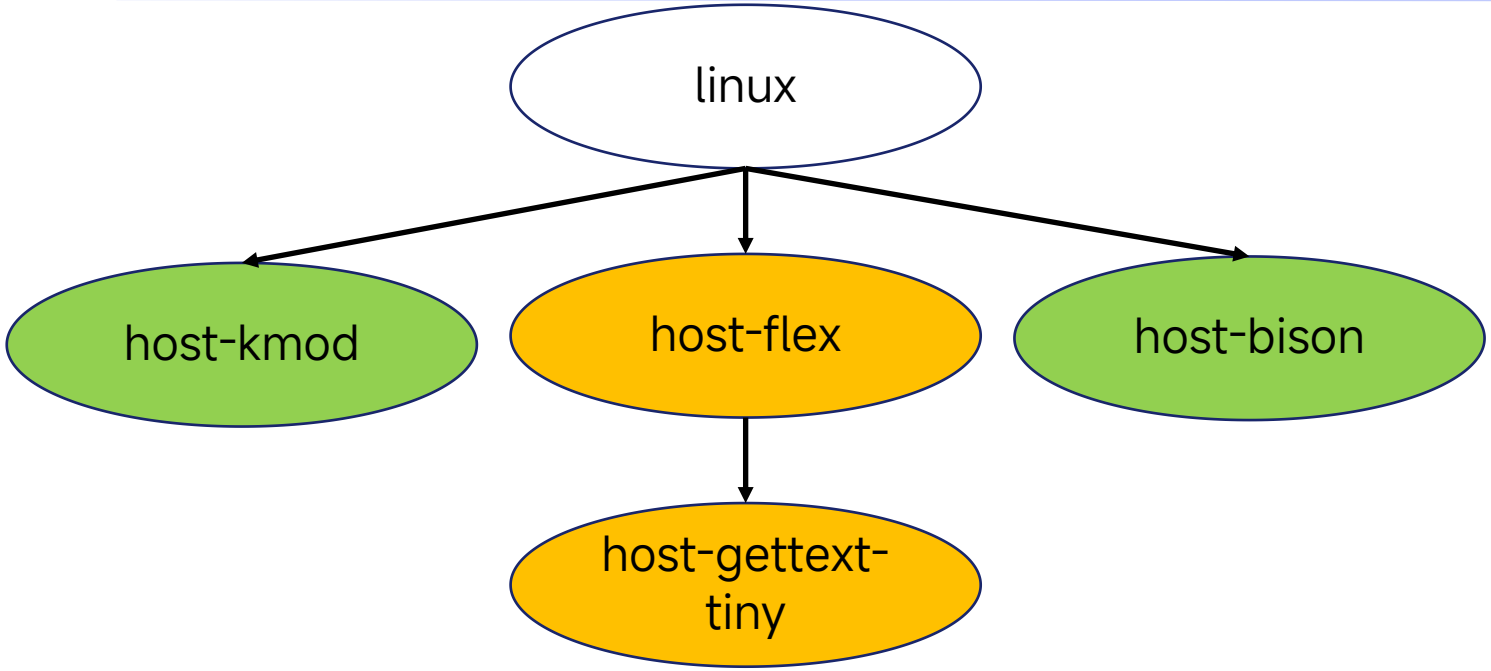
构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR}"
```

安装 （install-host）

```
eval “${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install -C
${PKGBUILD_DIR}”
mkdir -p ${HOST_DIR}/sbin/
ln -sf ../bin/kmod ${HOST_DIR}/sbin/depmod
```

制作 Linux 内核



Gettext-tiny - 简介

- GNU Gettext 是一个完整的软件国际化框架，支持将软件的文本输出翻译成多种语言。
- Gettext-tiny 是在不需要完整支持多语言环境下针对 GNU Gettext 的一个轻量级替代品。
- 我们这里因为不考虑支持多语言，所以采用 Gettext-tiny，而不是 Gettext，既保证了那些依赖于 Gettext 接口的软件包（如 Flex）可以顺利编译，同时又可以最小化系统开销并加快构建速度。

Gettext-tiny - 制作步骤

无需额外配置

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR} ${HOST_CONFIGURE_OPTS} prefix=${HOST_DIR} CFLAGS=\"-O2 -I${HOST_DIR}/include -fPIC\" LIBINTL=NONE"
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR} ${HOST_CONFIGURE_OPTS} prefix=${HOST_DIR} LIBINTL=NONE install"
```

Flex - 简介

https://www.gnu.org/software/flex/

- Flex（Fast Lexical Analyzer Generator）是一个用于快速生成词法分析器的工具。生成的词法分析器将通过扫描输入字符流，根据定义的规则将其切分成一个个有意义的词法单元。
- 使用 Flex 时我们编写一个后缀为 .l 的文件，用特定的规则定义词法单元（Tokens）、正则表达式模式以及关联的动作（C代码）
- Flex 读取这个 .l 文件，并将其转换成一个完整的、用 C 语言编写的词法分析器的源文件。
- Flex 通常与 Bison 配对使用。Bison 是一个语法分析器生成器，接受后缀为 .y 的文件。
- 在 Linux 中的应用场景：Kconfig、DTC

Flex - 制作步骤

配置

```
eval "AUTOPOINT=${HOST_DIR}/bin/autopoint ${AUTORECONF_OPTS}
${HOST_DIR}/bin/autoreconf -f -i"

patch_libtool ${PKGBUILD_DIR}

eval "${HOST_CONFIGURE_OPTS}  CONFIG_SITE=/dev/null ./configure --
prefix=\"${HOST_DIR}\" --sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\"
--enable-shared --disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --
disable-docs --disable-documentation --disable-debug --with-xmlto=no --with-fop=no --
disable-nls --disable-dependency-tracking  --disable-doc"
```

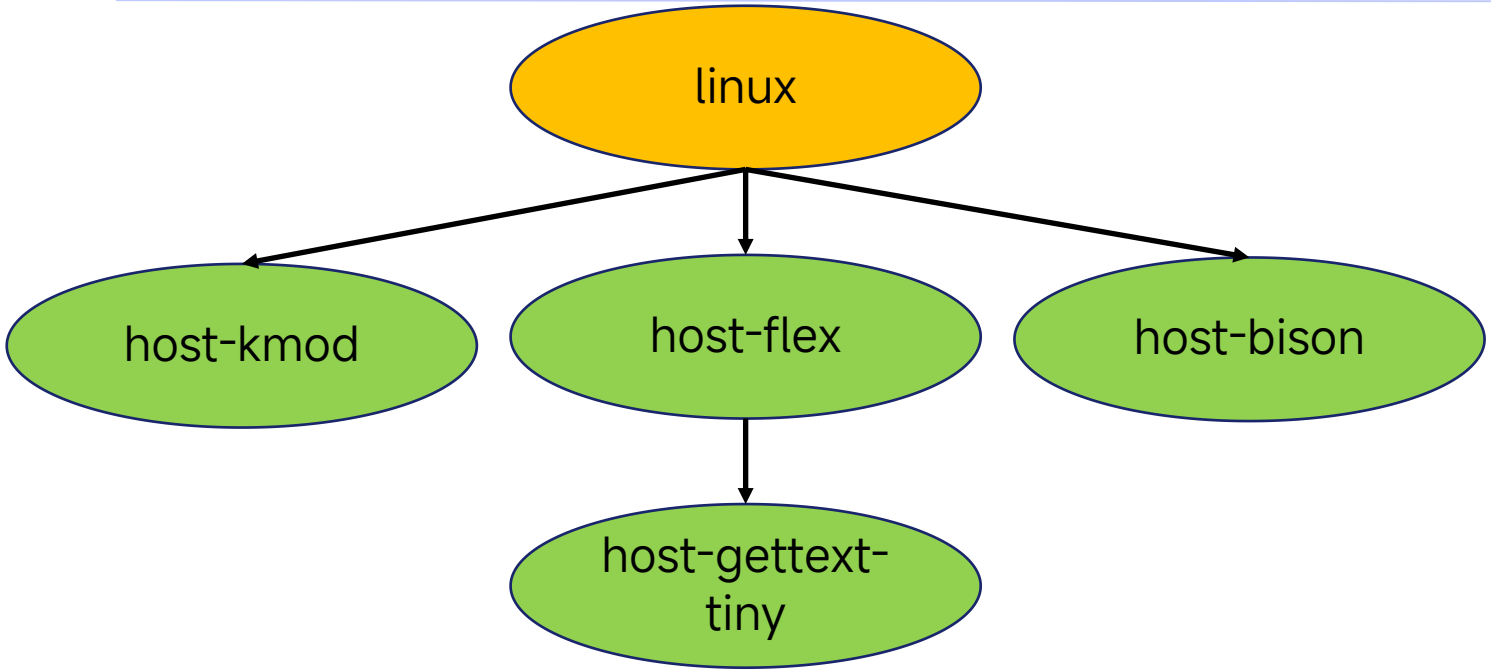
构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS}  -C ${PKGBUILD_DIR}"
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install -C ${PKGBUILD_DIR}"
```

制作 Linux 内核



制作 Linux 内核

配置

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR} ARCH=riscv KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR} CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1 DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 HOSTCC=\"/usr/bin/gcc\" defconfig"

kconfig_enable_option CONFIG_KERNEL_GZIP
kconfig_enable_option CONFIG_CPU_LITTLE_ENDIAN
kconfig_enable_option CONFIG_DEVTMPFS
kconfig_enable_option CONFIG_DEVTMPFS_MOUNT
kconfig_disable_option CONFIG_KERNEL_LZ4
kconfig_disable_option CONFIG_KERNEL_LZMA
kconfig_disable_option CONFIG_KERNEL_LZO
kconfig_disable_option CONFIG_KERNEL_XZ
kconfig_disable_option CONFIG_KERNEL_ZSTD
kconfig_disable_option CONFIG_KERNEL_UNCOMPRESSED
kconfig_disable_option CONFIG_GCC_PLUGINS
kconfig_disable_option CONFIG_WERROR

eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR} ARCH=riscv KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR} CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1 DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 HOSTCC=\"/usr/bin/gcc\" olddefconfig"
```

制作 Linux 内核

配置

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR} ARCH=riscv KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR} CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1 DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 HOSTCC=\"/usr/bin/gcc\" defconfig"

kconfig_enable_opt  CONFIG_KERNEL_GZIP
kconfig_enable_option CO  CPU_LITTLE
kconfig_enable_option CONFIG_D
kconfig_enable_option CONFIG_DEVTM
kconfig_disable_option CONFIG_KERNEL_L
kconfig_disable_option CONFIG_KERNEL_L
kconfig_disable_option CONFIG_KERNEL_L
kconfig_disable_option CONFIG_KERNEL_X
kconfig_disable_option CONFIG_KERNEL_ZSTD
kconfig_disable_option CONFIG_KERNEL_UNCOMPRESSED
kconfig_disable_option CONFIG_GCC_PLUGINS
kconfig_disable_option CONFIG_WERROR

eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR} ARCH=riscv KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR} CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1 DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 HOSTCC=\"/usr/bin/gcc\" olddefconfig"
```

先通过 make defconfig 利用默认配置获得一个干净的、可工作的基础配置 .config。

制作 Linux 内核

配置

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR} ARCH=riscv KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR} CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1 DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 HOSTCC=\"/usr/bin/gcc\" defconfig"

kconfig_enable_option CONFIG_KERNEL_GZIP
kconfig_enable_option CONFIG_CPU_LITTLE_ENDIAN
kconfig_enable_option CONFIG_DEVTMPFS
kconfig_enable_option CONFIG_DEVTMPFS_MOUNT
kconfig_disable_option CONFIG_KERNEL_LZ4
kconfig_disable_option CONFIG_KERNEL_LZMA
kconfig_disable_option CONFIG_KERNEL_LZO
kconfig_disable_option CONFIG_KERNEL_XZ
kconfig_disable_option CONFIG_KERNEL_ZSTD
kconfig_disable_option CONFIG_KERNEL_UNCOMPRESSED
kconfig_disable_option CONFIG_GCC_PLUGINS
kconfig_disable_option CONFIG_WERROR

eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR} ARCH=riscv KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR} CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1 DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 HOSTCC=\"/usr/bin/gcc\" olddefconfig"
```

手动修改 .config 中的一些关键配置

- kconfig_enable_option：“CONFIG_XXX=y”
- kconfig_disable_option：“# CONFIG_XXX is not set”

制作 Linux 内核

配置

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR} ARCH=riscv KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR} CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1 DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 HOSTCC=\"/usr/bin/gcc\" defconfig"

kconfig_enable_option CONFIG_KERNEL_GZ
kconfig_enable_option CONFIG_CPU_LITT
kconfig_enable_option CONFIG_DEVTMPF
kconfig_enable_option CONFIG_DEVTMPF
kconfig_disable_option CONFIG_KERNEL_
kconfig_disable_option CONFIG_KERNEL_
kconfig_disable_option CONFIG_KERNEL_
kconfig_disable_option CONFIG_KERNEL_
kconfig_disable_option CONFIG_KERNEL_ZS
kconfig_disable_option CONFIG_KERNEL_UNCO
kconfig_disable_option CONFIG_GCC_PLU
kconfig_disable_option CONFIG_WERR

eval "${HOST_MAKE_ENV} /us     make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR} ARCH=riscv KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR} CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1 DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 HOSTCC=\"/usr/bin/gcc\" olddefconfig"
```

通过 make olddefconfig 读取已有的 .config 文件，保留所有已设置的配置选项，同时将 .config 升级到当前使用的最新的内核状态，如果出现新的配置选项时，会自动使用它们的默认值，而不会像 make oldconfig 那样交互式地询问。

制作 Linux 内核

配置

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR} ARCH=riscv KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR} CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1 DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 HOSTCC=\"/usr/bin/gcc\" defconfig"
kconfig_enable_op
kconfig_enable_
kconfig_enable_
kconfig_enable_
kconfig_disable_
kconfig_disable_
kconfig_disable_
kconfig_disable_
kconfig_disable_
kconfig_disable_
kconfig_disable_
kconfig_disable_
eval "${HOST_M
attribute-alias\"
REGENERATE_P
olddefconfig"
```

选项	含义/用途
ARCH=riscv	指定内核支持的 ARCH
KCFLAGS="-Wno-attribute-alias"	给编译内核 C 文件的每个编译命令添加额外的编译选项
INSTALL_MOD_PATH=${TARGET_DIR}	用于指定内核模块安装（make modules_install）的目标路径前缀（基础目录）
CROSS_COMPILE="${CROSS_COMPILE}"	指定交叉编译器的前缀
WERROR=0	将警告（warning）视为普通信息而非错误，允许编译继续。
REGENERATE_PARSERS=1	强制重新生成内核配置的语法解析器文件
DEPMOD=${HOST_DIR}/sbin/depmod	指定 depmod 工具（生成模块依赖关系）的路径depmod 是 kmod 包的一部分
INSTALL_MOD_STRIP=1	安装模块时自动 strip 调试符号，减少模块文件大小。
HOSTCC="/usr/bin/gcc"	指定在构建内核过程中涉及构建在本地主机上运行的软件时所使用的编译器，注意和 CC 区别（CC 用于构建内核本身）

制作 Linux 内核

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} HOSTCC=\"/usr/bin/gcc -O2 -
isystem ${HOST_DIR}/include -L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib\" ARCH=riscv
KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR}
CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1
DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 -C ${PKGBUILD_DIR} all"

eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} HOSTCC=\"/usr/bin/gcc -O2 -
isystem ${HOST_DIR}/include -L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib\" ARCH=riscv
KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR}
CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1
DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 -C ${PKGBUILD_DIR} Image"
```

制作 Linux 内核

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} HOSTCC=\"/usr/bin/gcc -O2 -
isystem ${HOST_DIR}/include -L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib\" ARCH=riscv
KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR}
CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1
DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 -C ${PKGBUILD_DIR} all"

eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} HOSTCC=\"/usr/bi    -O2 -
isystem ${HOST_DIR}/include -L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib\" A    scv
KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR}
CROSS_COMPILE=\"${CROSS_COMPILE}\" WERR
DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_
```

make all 将生成：

- vmlinux：**未压缩**的 **ELF** 格式的可执行文件，是编译过程最终生成的“原始”内核映像（**包含调试信息，不可直接引导**）。
- 所有启用的模块(*.ko)

制作 Linux 内核

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} HOSTCC=\"/usr/bin/gcc -O2 -isystem ${HOST_DIR}/include -L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib\" ARCH=riscv KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR} CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1 DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 -C ${PKGBUILD_DIR} all"

eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} HOSTCC=\"/usr/bin/gcc -O2 -isystem ${HOST_DIR}/include -L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib\" ARCH=riscv KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR} CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1 DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 -C ${PKGBUILD_DIR} Image"
```

- make image 将针对不同的架构，在 vmlinux 的基础上经过进一步处理，生成可引导的镜像文件，用于制作发行版和安装内核：
- 针对 RISC-V 架构，make image 将生成 Image 和 Image.gz （Image 的压缩版本，前提是配置启用了 CONFIG_KERNEL_GZIP 等压缩选项）

制作 Linux 内核

安装 （Install-target）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS}  HOSTCC=\"/usr/bin/gcc -O2 -isystem ${HOST_DIR}/include -L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib\" ARCH=riscv KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR} CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1 DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 -C ${PKGBUILD_DIR} modules_install"
```

安装 （Install-image）

```
/usr/bin/install -m 0644 -D ${PKGBUILD_DIR}/arch/riscv/boot/Image ${IMAGES_DIR}/Image
```

制作 Linux 内核

安装 （Install-target）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS}  HOSTCC=\"/usr/bin/gcc -O2 -
isystem ${HOST_DIR}/include -L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib\" ARCH=riscv
KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR}
CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1
DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 -C ${PKGBUILD_DIR}
modules_install"
```

安装 （Install-image）

```
/usr/bin/install -m 0644 -D ${PKG                    /arch/riscv/boot/Image ${IMAGES_DIR}/Image
```

将内核模块（*.ko）安装到 ${TARGET_DIR}/lib/modules/6.12.47 下。

制作 Linux 内核

安装 （Install-target）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS}  HOSTCC=\"/usr/bin/gcc -O2 -isystem ${HOST_DIR}/include -L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib\" ARCH=riscv KCFLAGS=\"-Wno-attribute-alias\" INSTALL_MOD_PATH=${TARGET_DIR} CROSS_COMPILE=\"${CROSS_COMPILE}\" WERROR=0 REGENERATE_PARSERS=1 DEPMOD=${HOST_DIR}/sbin/depmod INSTALL_MOD_STRIP=1 -C ${PKGBUILD_DIR} modules_install"
```

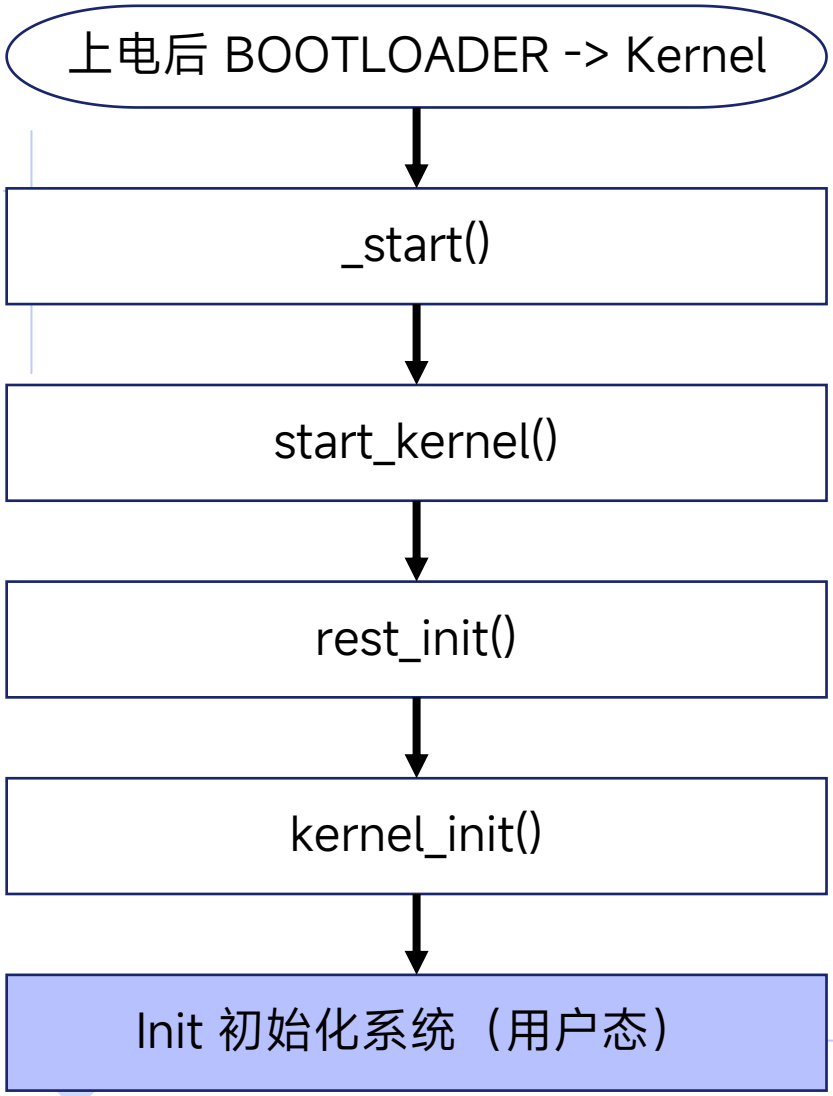
安装 （Install-image）

```
/usr/bin/install -m 0644 -D ${PKGBUILD_DIR}/arch/riscv/boot/Image ${IMAGES_DIR}/Image
```

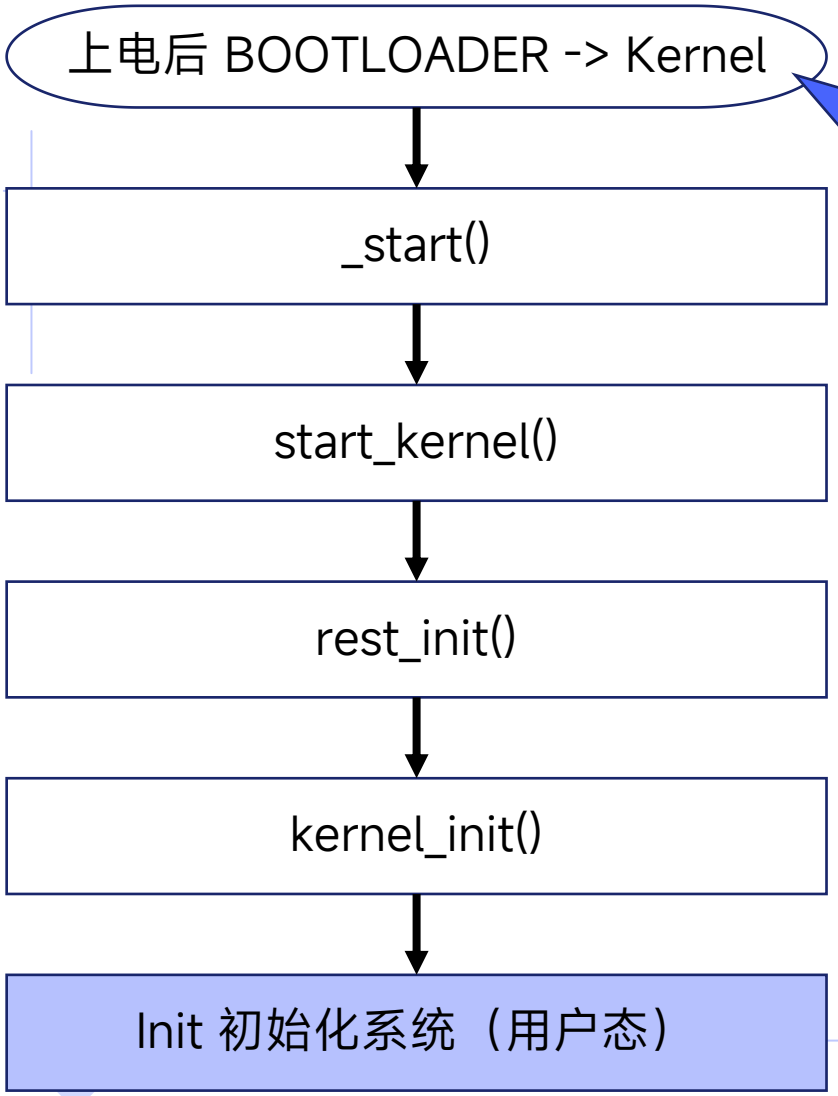
03

从内核态到用户态

内核启动到第一个用户进程 init



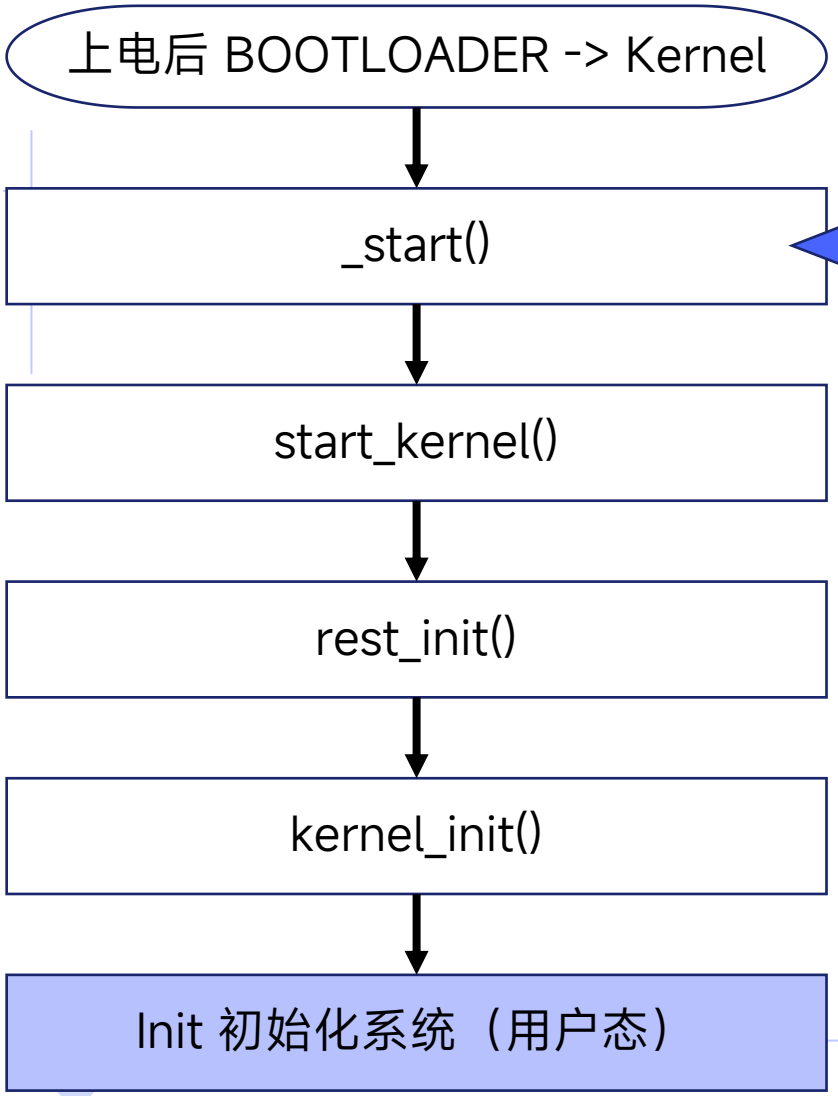
内核启动到第一个用户进程 init



BOOTLOADER 移交控制权

- BOOTLOADER（如 GRUB、U-Boot）将内核镜像（Image 或压缩的 zImage/uImage）和 initramfs（可选）加载到内存中指定的位置。它还会设置好一些必要的硬件参数（如设备树 dtb 在内存中的地址），然后跳转到内核的入口点（ _start）。
- CPU 处于 S-mode，禁用中断，内存管理单元（MMU）尚未开启。
- 在 QEMU riscv64 virt 方式下，可能跳过 BOOTLOADER，直接从 OpenSBI 跳转到 Linux 内核。

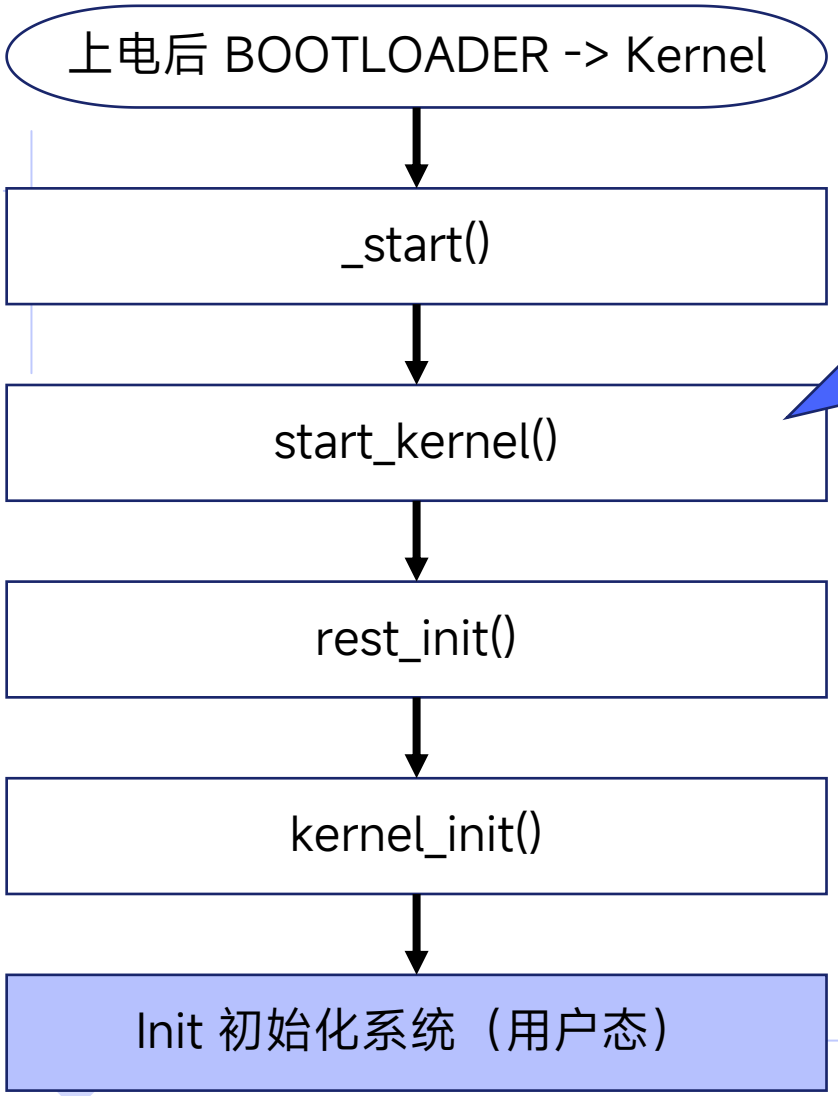
内核启动到第一个用户进程 init



内核早期初始化（汇编形式），包括：

- 解压内核： 如果内核是压缩的。
- 底层初始化：最基本的 CPU 设置，为开启 MMU 做准备（建立临时页表等）。
- 开启 MMU： 这是一个关键转折点。开启后，内核和后续所有程序将运行在虚拟地址空间。
- 跳转到 C 语言世界：设置好 C 语言运行环境（初始化栈、清除 BSS 段等），然后跳转到体系结构无关的 C 语言主函数 start_kernel()。

内核启动到第一个用户进程 init

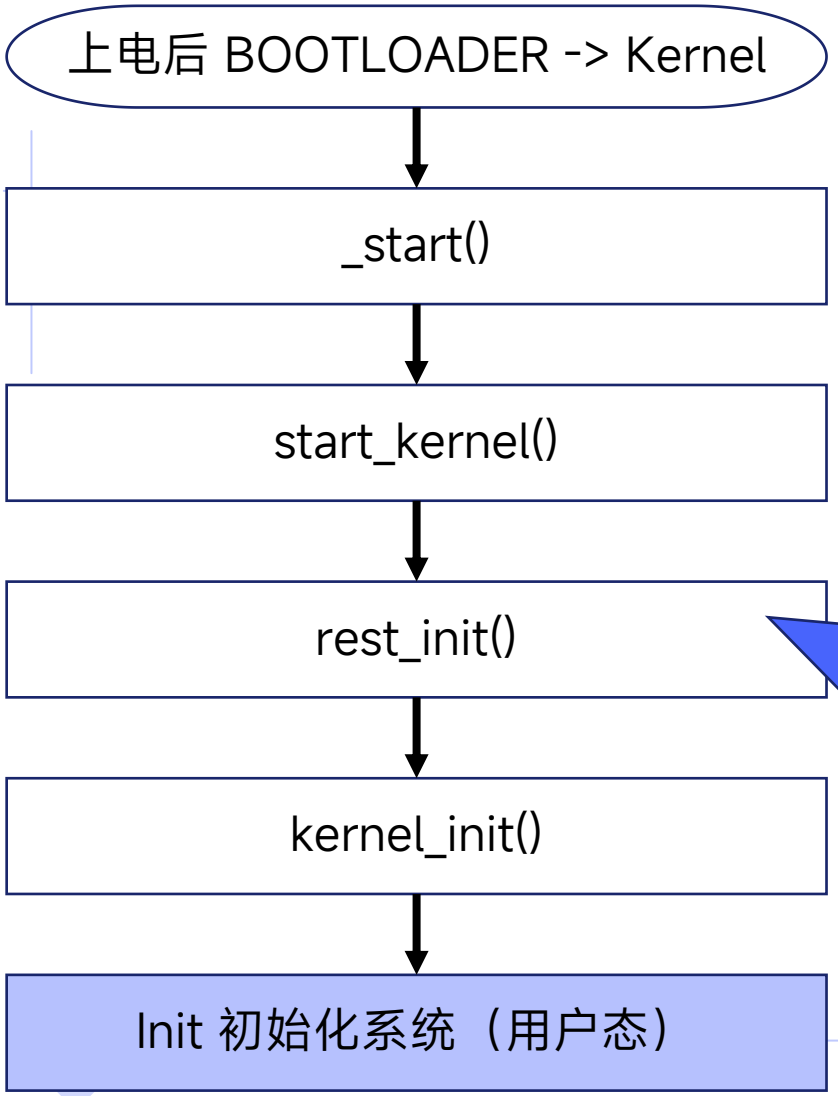


内核初始化（C 函数形式），调用一系列初始化子系统的函数，主要包括：

- 设置异常/中断处理
- 内存管理子系统初始化
- 进程调度器初始化
- 等等，包括早期控制台初始化打印出著名的内核启动日志开头：“Linux version ...”

最后调用的函数 rest_init()。至此，内核的核心基础设施已就绪。

内核启动到第一个用户进程 init

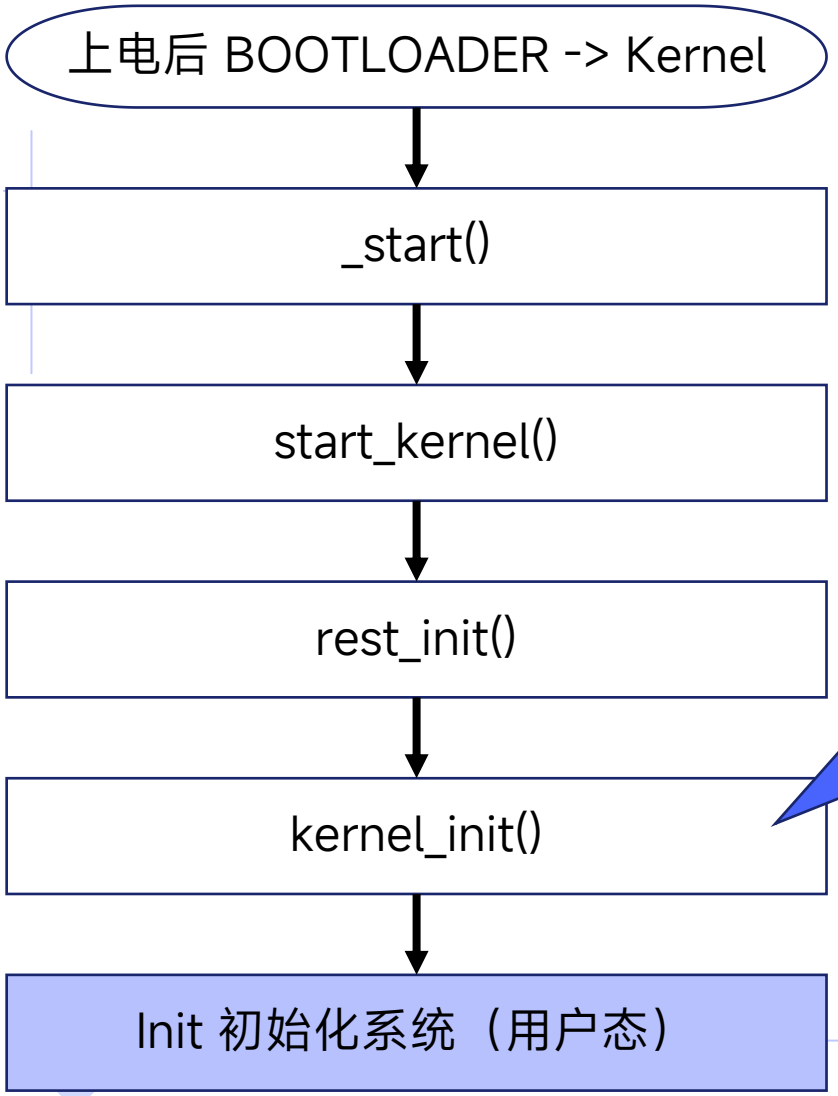


准备进入多任务：

- 创建内核线程 kernel_init（LWP ID = 1）：该内核线程将会转化成用户空间 `init` 进程。
- 创建内核线程 kthreadd （LWP ID = 2），这是内核的守护进程管理线程，负责调度其他内核线程。
- 将当前执行线程（start_kernel）转化为 idle 线程（LWP ID = 0），当 CPU 无事可做时，就运行它（WFI 低功耗睡眠）。
- 调用 schedule_preempt_disabled() 启用任务调度。
- 调用 cpu_startup_entry() 进入 idle 循环。

内核进入真正的多任务运行状态。kernel_init 和 kthreadd 线程在就绪队列中等待被调度。

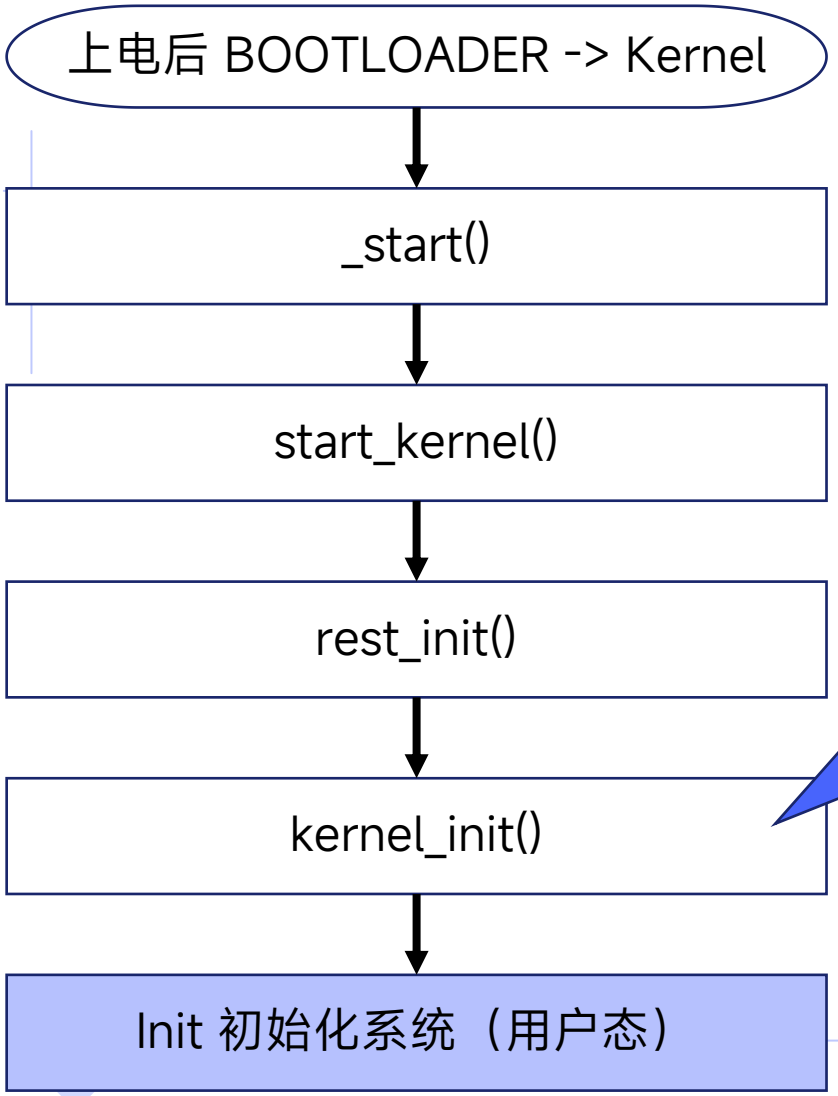
内核启动到第一个用户进程 init



内核线程 kernel_init 被调度执行，准备用户空间：

- 等待 kthreadd 启动完成。
- 初始化内核剩余子系统（驱动程序框架、文件系统等）。
- 尝试挂载根文件系统
- 清理，释放内存。
- 执行根文件系统中的用户程序 init

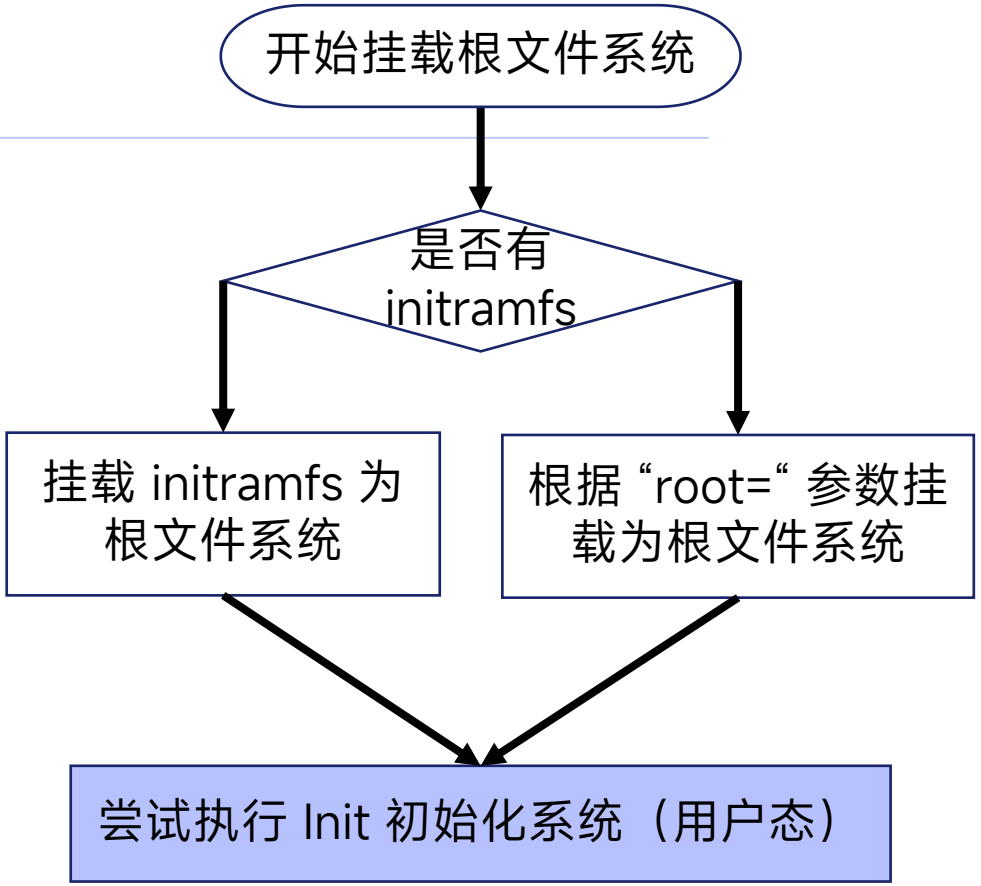
内核启动到第一个用户进程 init



内核线程 kernel_init 被调度执行，准备用户空间：

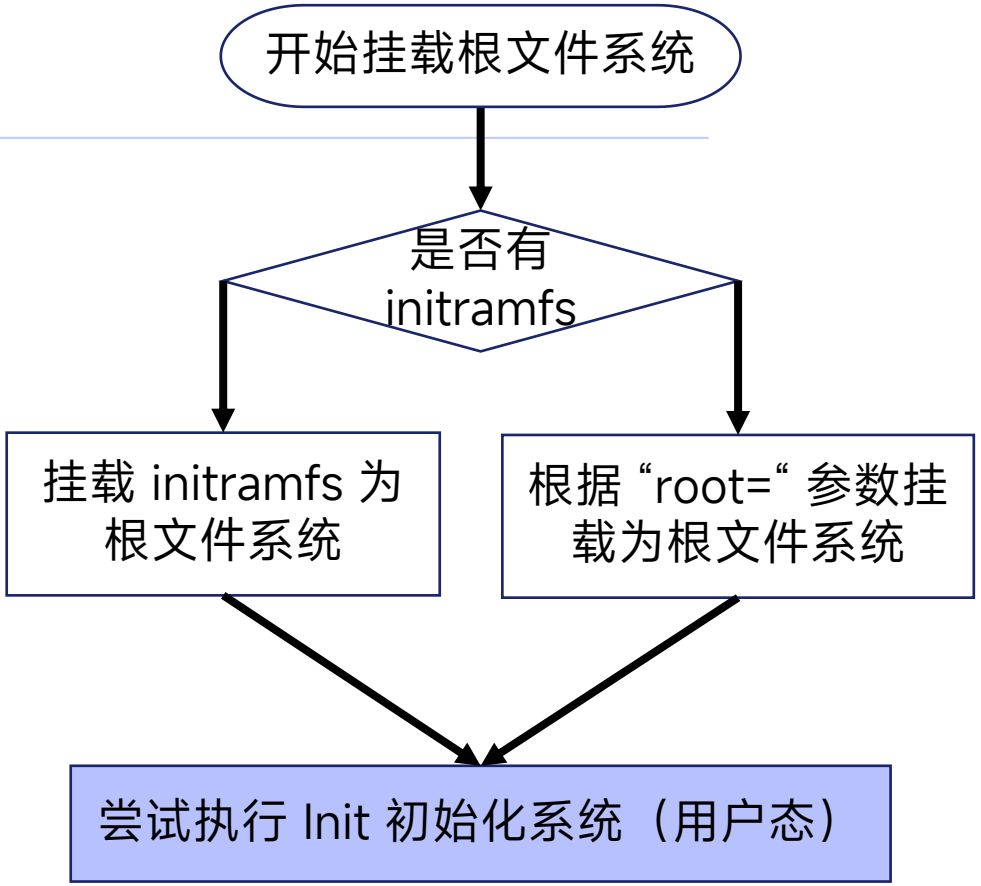
- 等待 kthreadd 启动完成。
- 初始化内核剩余子系统（驱动程序框架、文件系统等）。
- 尝试挂载根文件系统。
- 清理，释放内存。
- 执行根文件系统中的用户程序 init。

内核挂载根文件系统



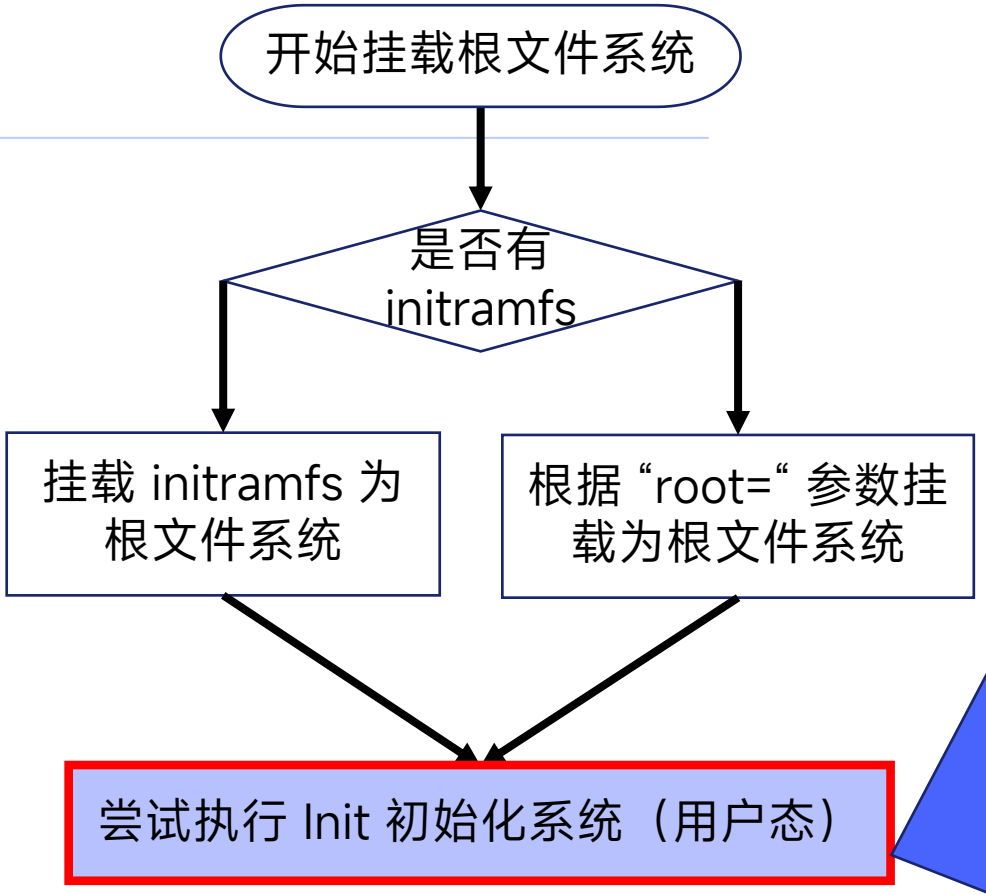
- Initramfs 是一个在系统启动早期、由内核直接加载到内存中的小型临时根文件系统。
- 引入 initramfs 的目的是为了解决内核启动时的以下关键矛盾：
  - 内核启动系统时需要找到并运行第一个用户程序 init，而这个用户程序本身存放在一个文件系统中。
  - 启动阶段的文件系统类型变得日趋复杂（如 RAID 阵列、LVM 卷、加密硬盘、网络存储等），仅凭 “root=” 命令行参数不足以满足内核处理和挂载这些复杂的文件系统类型。
- Initramfs 可以作为系统启动过程中内核与最终系统之间的一个 “跳板”，保持内核的精简同时提供极大的灵活性。

内核挂载根文件系统



- Initramfs 通常采用 CPIO（“Copy In, Copy Out”）格式的压缩包，由内核解压到基于内存的 tmpfs 文件系统中。
- Initramfs 作为一个文件系统，内核将其解压后会默认执行 “/init”, 也可以通过 “rdinit=” 参数指定其他的 init 程序的路径。
- 进一步的最终文件系统的复杂性以及挂载问题由 initramfs 中用户态的 init 以及后继程序处理，内核一旦完成 init 程序的 exec 后就完成了它在启动中的使命。
- 如果系统无需挂载最终的根文件系统，也可以停留在 initfamfs 中，但注意由于 initramfs 是一个基于内存的文件系统，不支持持久性。

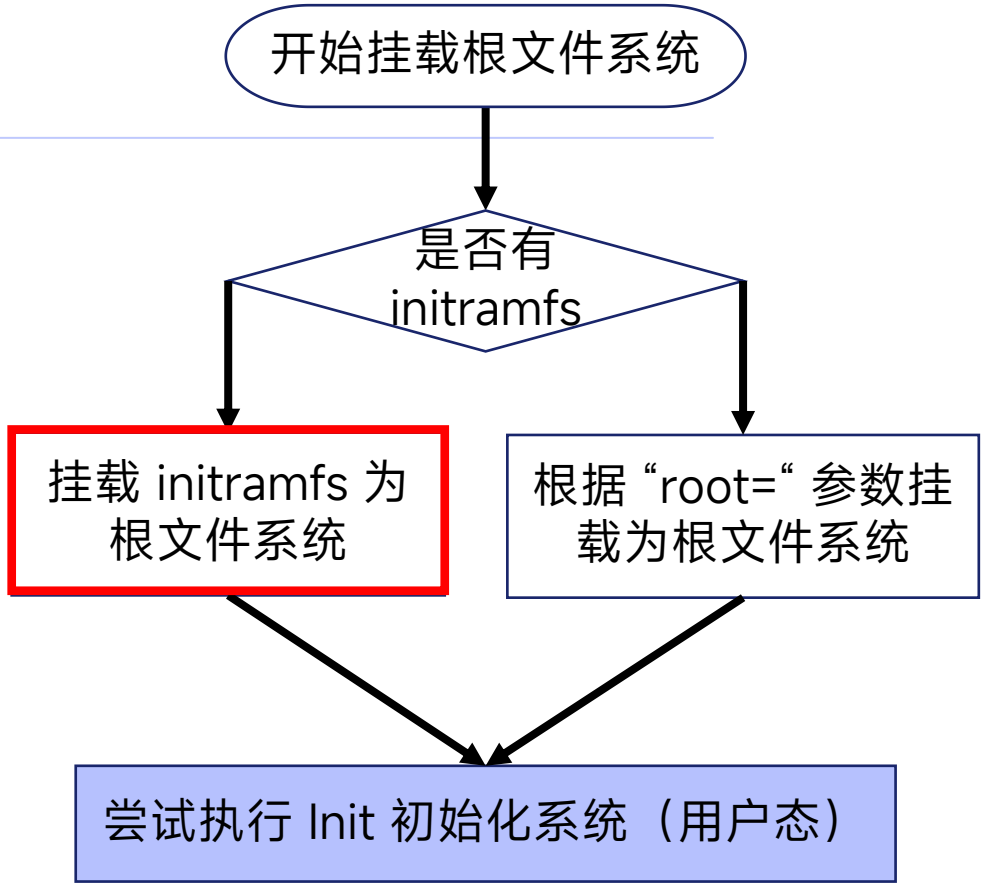
内核挂载根文件系统



kernel_init() @init/main.c (6.12.47)：

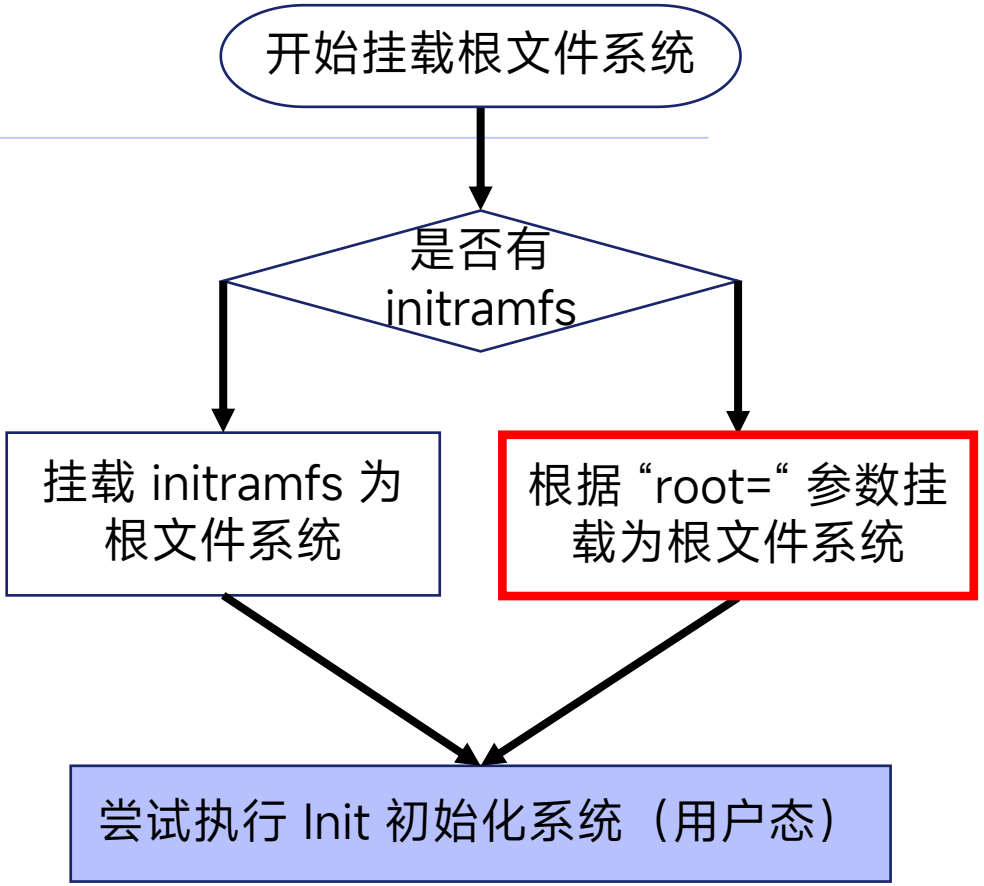
- 如果有 initramfs 则挂载 initramfs 为根文件系统并尝试执行 initramfs 下的 init，如果没有在命令行参数中设置 “rdinit=” 则默认是 “/init”
- 如果没有 initramfs，则根据 “root=” 指定的设备将其挂载为根文件系统，并按如下顺序依次尝试执行 init：
  - 命令行中是否有通过 “init=” 指定 init 路径，
  - 是否内核配置中 CONFIG_DEFAULT_INIT 设置了 init 路径
  - 如果上面都不成功则按照 “/sbin/init” -> “/etc/init” -> “/bin/init” -> “/bin/sh” 的顺序依次尝试
- 最后都不成功，则 panic

内核挂载根文件系统



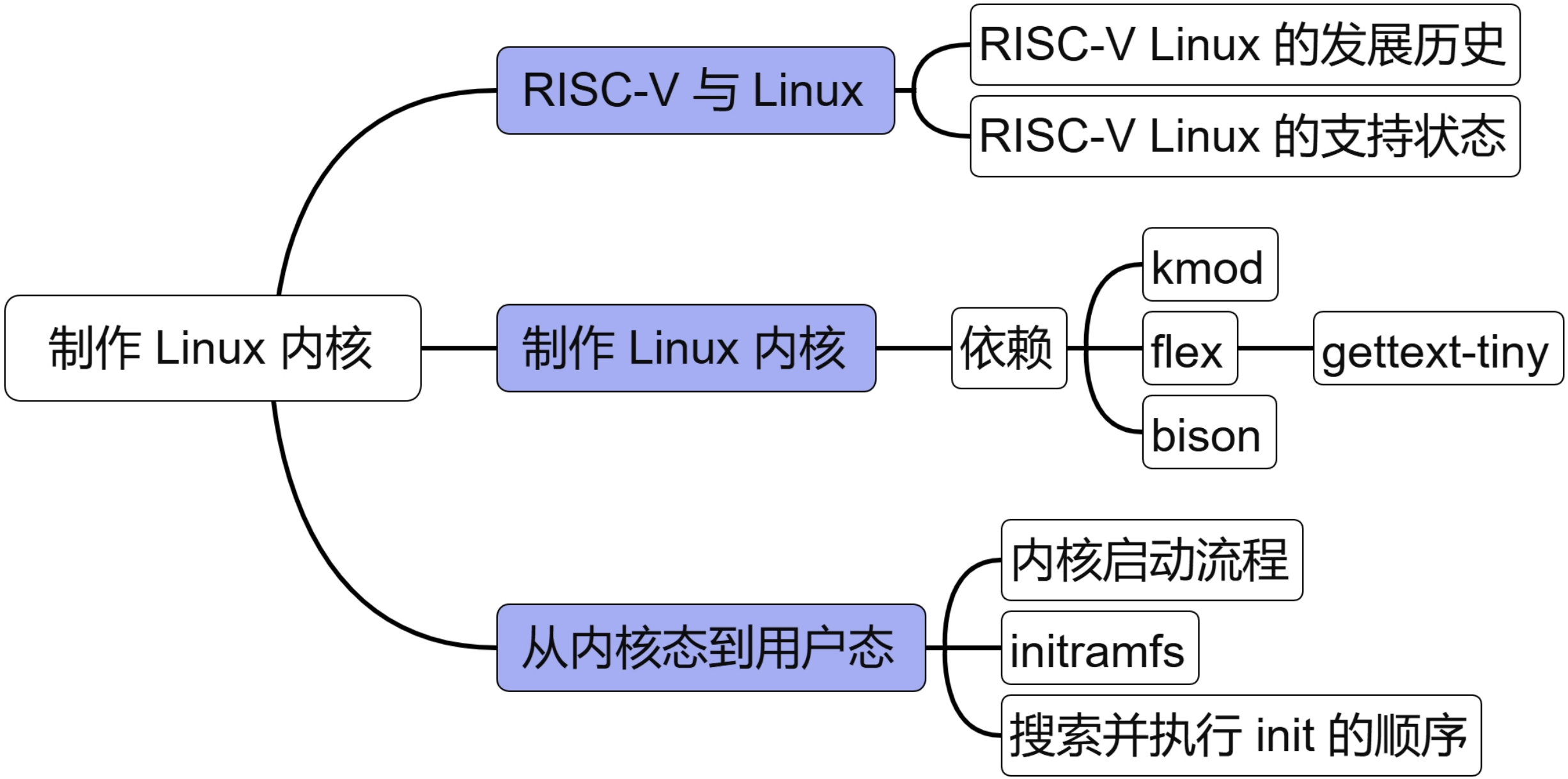

```
qemu-system-riscv64 \
-M virt -m 256M -nographic \
-bios fw_jump.bin \
-kernel Image \
-initrd initrd.img \
-append "nokaslr rdinit=/sbin/init console=ttyS0“ \
-netdev user,id=net0 -device virtio-net-
device,netdev=net0
```

内核挂载根文件系统




```
qemu-system-riscv64 -M virt -m 256M -nographic \
-bios fw_jump.bin \
-kernel Image \
-drive file=rootfs.ext2,format=raw,id=hd0 -device virtio-blk-device,drive=hd0 \
-append "nokaslr root=/dev/vda rw console=ttyS0" \
-netdev user,id=net0 -device virtio-net-device,netdev=net0
```

本章总结


制作 Linux 内核
RISC-V 与 Linux
RISC-V Linux 的发展历史
RISC-V Linux 的支持状态
制作 Linux 内核
制作 Linux 内核
依赖
kmod
flex
gettext-tiny
bison
从内核态到用户态
内核启动流程
initramfs
搜索并执行 init 的顺序

谢谢