五 Linux 内核编译与运行

实验目的

- 1. 掌握 Linux 内核镜像编译方法:
- 2. 熟悉 BusyBox、QEMU 等工具使用。

实验环境

- 1. 安装有 Linux 操作系统并连接互联网的计算机;
- 2. Linux 内核包:
- 3. BusyBox 工具包;
- 4. QEMU 模拟器。

实验步骤

常见 Linux 内核编译有两种方式,一是直接在 Linux 系统上编译得到二进制文件,并对原有 Linux 内核进行替换,即更换 Linux 内核,此方法可能因新内核有 bug 导致系统奔溃,且难以返回原版本内核而不得不重装系统;第二种方法则是在模拟器中运行新的 Linux 内核,以避免对系统内核的修改。

BusyBox 是一个集成了三百多个最常用 Linux 命令和工具的软件,因为单独的 Linux 内核无任何用于用户交互的 UI,所以需要通过其它工具与新编译的 Linux 内核交互。

QEMU 是以 GPL 许可证分发源码的模拟处理器,可用于模拟常见的硬件平台, 常用于在 Linux 系统中建立虚拟机。

本实验演示在 ubuntu 20.04 操作系统环境下编译 ARM Linux 内核。过程中主要是用交叉编译工具链 gcc-arm-linux-gnueabi 编译系统源码,并使用 QEMU 软件仿真硬件平台测试对象系统

本实验建议使用 root 用户操作。

1. 工具准备

Busybox 需手动下载安装, QEMU 等其他工具可在线安装。

2. 环境配置

Linux 内核编译环境需要大量软件包,可提前直接在线安装,或在内核编译过程中安装,若缺少安装包,内核编译过程中会提示缺失错误。以下是部分需要

的软件包,其中部分相同功能的软件包在不同的 Linux 版本下会以不同的名字存在。

apt-get install gcc qemu qemu-system-arm gcc-arm-linux-gnueabi libncurses5-dev build-essential flex bison bc

3. 编译最小文件系统

拷贝 busybox 至根目录,解压 busybox,进入目录并编译:

tar - jxvf busybox-1.31.1.tar.bz2

cd busybox-1.31.1

export ARCH=arm

export CROSS COMPILE=arm-linux-gnueabi-

make menuconfig

以上内容中,"export"后是指定交叉编译工具链,指定芯片框架为ARM。 如下图所示是图形化界面进行内核配置。

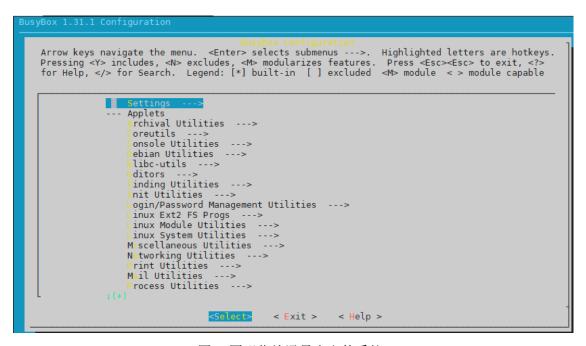


图 1 图形化编译最小文件系统

按照以下路径配置成静态编译。

Settings ───>

Build Options

[*]Build static binary (no shared libs)

如下图所示为配置界面

```
<Enter> selects submenus --->. Highlighted letters are hotkeys.
Arrow keys navigate the menu.
Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < > module capable
                             Support --show SCRIPT (NEW)
                    [*]
                              upport --install [-s] to install applet links at runtime (NEW)
                          on't use /usr (NEW)
                         Drop SUID state for most applets (NEW)
                   [*] Enable SUID configuration via /etc/busybox.conf (NEW)

[*] Suppress warning message if /etc/busybox.conf is not readable (NEW)

[] exec prefers applets (NEW)

(/proc/self/exe) Path to busybox executable (NEW)
                         Support NSA Security Enhanced Linux (NEW)
Clean up all memory before exiting (usually not needed) (NEW)
Support LOG_INFO level syslog messages (NEW)
                        Build Options
                      ] Force NOMMU build (NEW)
                         Cross compiler prefix (NEW)
Path to sysroot (NEW)
Additional CFLAGS (NEW)
                           dditional LDFLAGS (NEW)
                          Additional LDLIBS (NEW)
                                                 <Select>
                                                                   < Exit > < Help >
```

图 2 静态编译配置

配置完毕退出后继续完成编译:

make install

完成后会在目录中生成"_install"目录,本目录存放了编译好的文件系统需要的命令集合,如下图所示。

```
[yql@localhost busybox-1.31.1]$ ls
                                                                   Makefile.help
applets
                                                                   make_single_applets.sh
                                               INSTALL
                                                                                                size_single_applets.sh
sysklogd
AUTHORS
                                               LICENSE
                                                                   NOFORK_NOEXEC.lst
                            e2fsprogs
busvbox
                                               loginutils
                                                                   NOFORK_NOEXEC.sh
printutils
busýbox.links
busybox_unstripped
                                                                                                 TODO
                                                                                                TODO_unicode
util-linux
busybox_unstripped.map
busybox_unstripped.out
Config.in
                                               Makefile
                                                                   gemu multiarch_testing
                                               Makefile.custom
                                               Makefile.flags
                                                                   README
```

图 3 生成_install 目录

4. 编译内核

解压 Linux 内核文件包:

tar -xzvf linux-5.1.20. tar. gz

并将上一步骤中生成的"_install"目录拷贝至解压后的内核目录,进入 "install"目录,分别创建 etc、dev、mnt、etc/init.d 等目录。

mkdir etc

mkdir dev

mkdir mnt

mkdir -p etc/init.d

```
[yql@localhost busybox-1.31.1]$ cd /home/yql/lab/linux-5.1.20/_install/
[yql@localhost _install]$ sudo mkdir etc
[yql@localhost _install]$ sudo mkdir dev
[yql@localhost _install]$ sudo mkdir mnt
[yql@localhost _install]$ sudo mkdir -p etc/init.d/
```

图 4 创建目录

在 "_install/etc/init.d" 目录下新建 "rcS" 文件,并写入以下内容。

```
mkdir -p /proc
mkdir -p /tmp
mkdir -p /sys
mkdir -p /mnt
/bin/mount -a
mkdir -p /dev/pts
mount -t devpts devpts /dev/pts
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s
```

```
[yql@localhost _install]$ cd etc/init.d/
[yql@localhost init.d]$ sudo vi rcS
[yql@localhost init.d]$ sudo cat rcS
mkdir -p /proc
mkdir -p /tmp
mkdir -p /sys
mkdir -p /mnt
/bin/mount -a
mkdir -p /dev/pts
mount -t devpts devpts /dev/pts
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s
```

图 5 创建并编辑 rcS 文件

修改 rcS 文件权限,添加可执行权限,如下图所示为修改权限前后对比。

chmod 755 rcS

```
[yql@localhost init.d]$ ls -l
total 4
-rw-r--r--. 1 root root 172 May 3 04:25 rcS
[yql@localhost init.d]$
[yql@localhost init.d]$ sudo chmod 755 rcS
[yql@localhost init.d]$ ls -l
total 4
-rwxr-xr-x. 1 root root 172 May 3 04:25 rcS
```

图 6 修改 rcS 文件权限

在" install/etc"目录创建"fstab"文件,并写入以下内容:

```
proc /proc proc defaults 0 0
tmpfs /tmp tmpfs defaults 0 0
sysfs /sys sysfs defaults 0 0
tempfs /dev tmpfs defaults 0 0
debugfs /sys/kernel/debug debugfs defaults 0 0
     [yql@localhost etc]$ sudo vi fstab
     [yql@localhost etc]$ sudo cat fstab
     proc /proc proc defaults 0 0
     tmpfs /tmp tmpfs defaults 0 0
     sysfs /sys sysfs defaults 0 0
     tempfs /dev tmpfs defaults 0 0
     debugfs /sys/kernel/debug debugfs defaults 0 0
                  图 7 创建并编辑 fstab 文件
在" install/etc"目录创建"inittab"文件,并写入以下内容:
::sysinit:/etc/init.d/rcS
::respawn:-/bin/sh
::askfirst:-/bin/sh
::ctrlaltdel:/bin/umount -a -r
           [yql@localhost etc]$ sudo vi inittab
[yql@localhost etc]$ sudo cat inittab
           ::sysinit:/etc/init.d/rcS
           ::respawn:-/bin/sh
           ::askfirst:-/bin/sh
           ::ctrlaltdel:/bin/umount -a -r
                 图 8 创建并编辑 inittab 文件
在" install/dev"目录中创建如下设备节点。
mknod console c 5 1
mknod null c 1 3
[yql@localhost dev]$ sudo mknod console c 5 1
[vgl@localhost dev]$ sudo mknod null c 1 3
                       图 9 创建节点
完成上述设置后,在内核目录中编译内核。
export ARCH=arm
export CROSS COMPILE=arm-linux-gnueabi-
make vexpress defconfig
make menuconfig
```

```
root@bogon:/linux-5.1.20# export ARCH=arm
root@bogon:/linux-5.1.20# export CROSS_COMPILE='arm-linux-gnueabi-'
root@bogon:/linux-5.1.20# make vexpress_defconfig
```

图 10 编译内核

在图形化编译。

设置中,按照以下路径,在 initramfs source file 中填入"_install", General setup ---->

[*]Initial RAM filesystem and RAM disk (initramfs/initrd) support (install)Initramfs source file(s)

```
General setup

Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----).
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features.
Press <Esc>-Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [] excluded

<M> module < > module capable

[*] **System V IPC

[*] **POSIX Message Queues

[*] **Lenable process vm readv/writev syscalls

[*] **uselib* syscall

[*] **Juselib* syscall

[*] **Juselib* syscall

[*] **Lenable process vm readv/writev syscalls

[*] **Lenable process vm readv/writev syscalls

[*] **Lenable system --->

(**PU/Task time and stats accounting --->

RCU Subsystem --->

(**PU/Task time and stats accounting --->

RCU Subsystem --->

(**PU/Task time and stats accounting --->

RCU Subsystem --->

[*] **Lenable system and stats accounting --->

**Nemespaces support --->

[*] **Lenable access to .config through /proc/config.gz

(14) **Kernel log buffer size (16 ⇒ 64KB, 17 ⇒ 128KB)

[*] **Control Group support --->

-**Nemespaces support --->

[*] **Lenable system support of system sy
```

图 11 路径配置



图 12 路径配置

把 Default kernel command string 清空。下图中删除原有内容用Ctl+Backspace键。

Boot option -→

()Default kernel command string

```
.config - Linux/arm 4.0.1 Kernel Configuration

Boot options

Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----).

Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features.

Press <Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [] excluded

<M> module <> module capable

-*- Flattened Device Tree support

[*] Support for the traditional ATAGS boot data passing

[] Provide old way to pass kernel parameters

(0x0) Compressed ROM boot loader base address

(0x0) Compressed ROM boot loader BSS address

[] Use appended device tree blob to zImage (EXPERIMENTAL)

[] Meracl system call (EXPERIMENTAL)

[] Build kdump crash kernel (EXPERIMENTAL)

-*- Auto calculation of the decompressed kernel image address
```

图 13 路径配置

配置 memory split 为 "3G/1G user/kernel split",并打开高端内存。

Kernel features →

Memory split(3G/1G user/kernel split) →

[*] High Memory Support

```
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded
<M> module < > module capable
                   [*] Symmetric Multi-Processing
[*] Allow booting SMP kernel on uniprocessor systems (EXPERIMENTAL)
                            Support cpu topology definition
Multi-core scheduler support
                               SMT scheduler support
                        Architected timer support
                   [*] Multi-Cluster Power Management
[] big.LITTLE support (Experimental)
Memory split (3G/1G user/kernel split) --->
(8) Maximum number of CPUs (2-32)
-*- Support for hot-pluggable CPUs
                   [*] Support for the ARM Power State Coordination Interface (PSCI)
                   Preemption Model (No Forced Preemption (Server))
Timer frequency (100 Hz) --->

[] Compile the kernel in Thumb-2 mode
[*] Use the ARM EABI to compile the kernel
                             Allow old ABI binarie's to run with this kernel (EXPERIMENTAL)
                              llocate 2nd-level pagetables from highmem (NEW)
                    [*] Enable hardware performance counter support for perf events
                        Allow for balloon memory compaction/migration
                   [*] Allow for memory compaction
                                           < Exit > < Help > < Save > < Load >
```

图 14 路径配置

在内核目录下编译内核(此步骤时间较长)。

make bzImage ARCH=arm CROSS COMPILE=arm-linux-gnueabi-

编译完成后会有如下提示,并显示编译后内核的存储路径。

```
SHIPPED arch/arm/boot/compressed/bswapsdi2.S

AS arch/arm/boot/compressed/bswapsdi2.o

LD arch/arm/boot/compressed/vmlinux

OBJCOPY arch/arm/boot/zImage

Kernel: arch/arm/boot/zImage is ready

root@bogon:/linux-5.1.20# ls
```

图 15 内核编译成功

编译生成 dtb 文件。

make dtbs

编译完成后会有如下提示,并显示生成的 dtb 文件。

```
root@ubuntu:/linux-5.1.20# make dtbs
DTC arch/arm/boot/dts/vexpress-v2p-ca5s.dtb
DTC arch/arm/boot/dts/vexpress-v2p-ca9.dtb
DTC arch/arm/boot/dts/vexpress-v2p-ca15-tc1.dtb
DTC arch/arm/boot/dts/vexpress-v2p-ca15_a7.dtb
```

图 16 编译生成 dtb 文件

5. 运行 QEMU

如下图所示,输入 QEMU 启动命令,成功启动 QEMU,注意需指定 bzImage 路径,并注意使用当前命令与 bzImage 路径的关系(以下为一条命令)。

qemu-system-arm -M vexpress-a9 -m 256M -kernel arch/arm/boot/zImage -append

"rdinit=/linuxrc console=ttyAMAO loglevel=8" -dtb

arch/arm/boot/dts/vexpress-v2p-ca9.dtb -nographic

以上命令中参数含义如下;

-M: 指定硬件芯片框架

-m: 指定运行内存大小

-kernel: 指定运行的内核镜像

-dtb: 指定具体芯片的配置信息

-nographic: 指定不使用图形界面

```
Please press Enter to activate this console. random: fast init done

/ #
/ #
/ # pwd
/
/ # uname -a
Linux (none) 5.1.20 #1 SMP Sat May 9 00:56:29 CST 2020 armv7l GNU/Linux
/ #
/ #
```

图 16 启动 QEMU

实验内容

- 1、按照实验演示,完成内核编译模拟。
- 2、在QEMU中启动新内核。