实验记录

一、嵌入式交叉编译开发环境的构建与使用

1、查询本地操作系统是否具有交叉编译工具

```
root@ubuntu:yulong# ls
arm-linux-4.2.1  arm-linux-4.2.1.tar.gz
root@ubuntu:yulong# which arm-linux-gcc
/opt/arm-linux/bin/arm-linux-gcc
root@ubuntu:yulong#
```

2、通过查询，在操作系统中具有交叉编译工具，可直接进行交叉编译。

3、编辑源文件 hello.c，并使用本地编译和交叉编译两种形式进行编译生成可执行文件 hello_x86 和 hello_arm

```
root@ubuntu:yulong# vim hello.c
root@ubuntu:yulong# gcc hello.c -o hello_x86
root@ubuntu:yulong# arm-linux-gcc hello.c -o hello_arm
```

4、尝试对两个可执行文件执行，并观察输出结果

```
root@ubuntu:yulong# ./hello_x86
hello,yulong!
root@ubuntu:yulong# ./hello_arm
bash: ./hello_arm: 无法执行二进制文件
```

从输出结果看出，使用本地编译的可执行文件输出正常，而使用交叉编译的可执行文件不能在本地环境下执行。

二、编译嵌入式 Linux 系统的镜像文件 bootload

1、解压并安装编译工具 XSCALEV1

```
root@ubuntu:yulong# tar xzvf Boot-XSBase270.tar.gz
Boot-XSBase270/
Boot-XSBase270/config
Boot-XSBase270/include/
Boot-XSBase270/include/xsclkmgr.h
Boot-XSBase270/include/version.h
Boot-XSBase270/include/types.h
Boot-XSBase270/include/timers.h
Boot-XSBase270/include/time.h
Boot-XSBase270/include/systypes.h
Boot-XSBase270/include/string.h
Boot-XSBase270/include/stdio.h
Boot-XSBase270/include/stdarg.h
Boot-XSBase270/include/setup.h
Boot-XSBase270/include/serial.h
```

2、找到 src 目录和 include 目录，观察到用于第一阶段的.s 汇编文件和第二阶段的.c 和.h 文件

```
root@ubuntu:yulong_Boot-XSBase270# ls
boot  config  include  Makefile  src  utils
root@ubuntu:yulong_Boot-XSBase270# cd src
root@ubuntu:src# ls
boot.elf32   ctype.c      lan91c111.o  memsetup.S
boot.lds     ctype.o      linux.c      menu.c
boot.lds.in  fixgpio.o    linux.o      menu.o
bootp.c      fixgpio.S    main.c       network.c
bootp.o      flash.c      main.o       network.o
cmddebug.c   flash.o      Makefile     partition.
cmddebug.o   gpio.c       memcpy.o     partition.
command.c    gpio.o       memcpy.S     pxafb.c
command.o    lan91c111.c  memsetup.o   pxafb.o
root@ubuntu:src# cd ../include/
root@ubuntu:include# ls
audio.h      config.h      lan91c111.h  network.h
bitfield.h   ctype.h       memory.h     post.h
board        _desktop.ini  menu.h       pxareg.h
board.h      errors.h      misc.h       registers
byteorder.h  hardware.h    netdev.h     serial.h
command.h    io.h          netheader.h  setup.h
root@ubuntu:include#
```

3、使用 make clean 清除掉已经生成好的 boot，并使用 make 生成自己编辑好的 boot

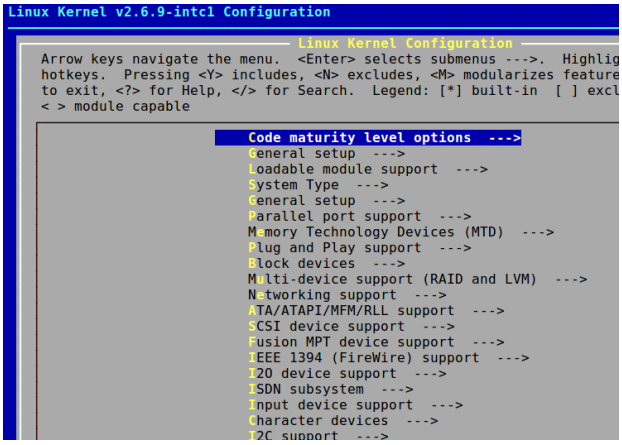
```
root@ubuntu:yulong_Boot-XSBase270# ls
boot  config  include  Makefile  src  utils
root@ubuntu:yulong_Boot-XSBase270# vim Makefile
root@ubuntu:yulong_Boot-XSBase270# ls
boot  config  include  Makefile  src  utils
root@ubuntu:yulong_Boot-XSBase270# make clean
In file included from contest.c:1:
include/stdio.h:7: warning: conflicting types for bu
root@ubuntu:yulong_Boot-XSBase270# make
In file included from contest.c:1:
include/stdio.h:7: warning: conflicting types for bu
In file included from contest.c:1:
include/stdio.h:7: warning: conflicting types for bu
In file included from contest.c:1:
include/stdio.h:7: warning: conflicting types for bu
compile start.S
compile memsetup.S
compile setup.c
setup.c:11: warning: parameter names (without types)
setup.c: In function `view_setup':
setup.c:95: warning: int format, uint32 arg (arg 3)
setup.c: In function `I2cTxEmpty':
setup.c:152: warning: unused variable `temp'
setup.c: In function `I2CWrite':
```

三、编译嵌入式 Linux 系统的镜像文件内核 zImage

1、对下载好的 Linux 内核进行解压缩

```
root@ubuntu:yulong# tar zxvf linux-2.6.9_eeliod.tar.gz
linux-2.6.9-eeliod/
linux-2.6.9-eeliod/Documentation/
linux-2.6.9-eeliod/Documentation/BK-usage/
linux-2.6.9-eeliod/Documentation/BK-usage/00-INDEX
linux-2.6.9-eeliod/Documentation/BK-usage/bk-kernel-howto.txt
linux-2.6.9-eeliod/Documentation/BK-usage/bk-make-sum
linux-2.6.9-eeliod/Documentation/BK-usage/bksend
linux-2.6.9-eeliod/Documentation/BK-usage/bz64wrap
linux-2.6.9-eeliod/Documentation/BK-usage/cpcset
linux-2.6.9-eeliod/Documentation/BK-usage/cset-to-linus
linux-2.6.9-eeliod/Documentation/BK-usage/csets-to-patches
linux-2.6.9-eeliod/Documentation/BK-usage/gcapatch
```

2、进入内核主目录，尝试使用 make menuconfig 打开菜单的图形界面



3、成功打开后进入 char 目录下，增加自定义功能

```
root@ubuntu:yulong_linux-2.6.9-eeliod# cd drivers/char/
root@ubuntu:char# ls
agp                  epca.c             mem.c
amiserial.c          epcaconfig.h       mem.o
applicom.c           epca.h             misc.c
applicom.h           esp.c              misc.o
built-in.o           fep.h              mmtimer.c
cd1865.h             ftape              moxa.c
ChangeLog            generic_nvram.c    mwave
console_macros.h     generic_serial.c   mxser.c
consolemap.c         genrtc.c           n_hdlc.c
consolemap_deftbl.c  hangcheck-timer.c  n_r3964.c
```

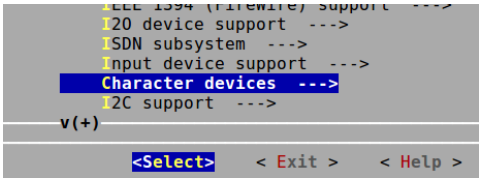
4、在 char 子目录下编辑 Kconfig 文件，增加自定义功能

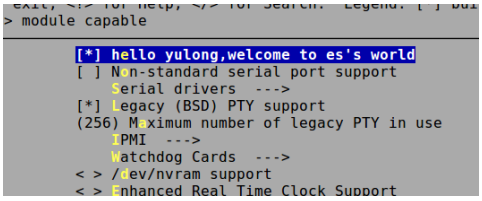
```
  38 config yulong_hello_modole
  39         bool "hello yulong,welcome to es's world"
  40
  41 config VT_CONSOLE
  42         bool "Support for console on virtual termina
  43         depends on VT
  44         default y
  45         ---help---
:wq
```

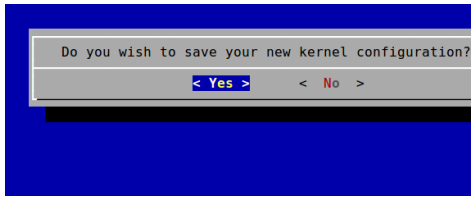
5、返回内核根目录，重新打开菜单图形界面

```
root@ubuntu:char# vim Kconfig
root@ubuntu:char# cd ../../
root@ubuntu:yulong_linux-2.6.9-eeliod# make menuconfig
```

6、找到新增自定义功能并选中







7、保存菜单更改结果后，在根目录打开.config 文件，看到新增自定义功能被赋值为 y，及该功能被选中，将写入内核中。

```
557 # Character devices
558 #
559 CONFIG_VT=y
560 CONFIG_yulong_hello_modole=y
561 CONFIG_VT_CONSOLE=y
562 CONFIG_HW_CONSOLE=y
563 # CONFIG_SERIAL_NONSTANDARD is not set
564
565 #
```

8、再次回到 char 子目录下，创建新文件 hello.c。

```
root@ubuntu:yulong_linux-2.6.9-eeliod# cd drivers/char/
root@ubuntu:char# cp XSB_EDR_8LED.c hello.c
```

9、将编译写入 Makefile 中与自定义功能名连接，即当自定义功能被选中是，hello.c 将会被编译，写入内核中。

```
11
12 obj-$(CONFIG_yulong_hello_modole) += hello.o
13 obj-$(CONFIG_LEGACY_PTYS)       += pty.o
14 obj-$(CONFIG_UNIX98_PTYS)       += pty.o
15 obj-y                           += misc.o
```

10、回到根目录下进行编译，编译过程中看到了 hello.o 的生成

```
root@ubuntu:yulong_linux-2.6.9-eeliod# cd drivers/char/
root@ubuntu:char# vim Makefile
root@ubuntu:char# cd ../../
root@ubuntu:yulong_linux-2.6.9-eeliod# vim Makefile
root@ubuntu:yulong_linux-2.6.9-eeliod# make
  CHK     include/linux/version.h
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf -s arch/arm/Kconfig
#
# using defaults found in .config
#
  SPLIT   include/linux/autoconf.h -> include/config/*
  SYMLINK include/asm-arm/arch -> include/asm-arm/arch-p
make[1]: “arch/arm/kernel/asm-offsets.s”是最新的。
make[1]: “include/asm-arm/mach-types.h”是最新的。
  CHK     include/linux/compile.h
  CC      drivers/char/hello.o
  LD      drivers/char/built-in.o
  CC [M]  drivers/char/XSB_EDR_8LED.o
```

11、使用 make zImage 对编译好的内核进行镜像文件的生成，生成过程中也看见了 hello.o 文件被写了进去，且在指定目录下找到了已经生成好的 zImage 文件

```
  CC      drivers/char/n_tty.o
  CC      drivers/char/tty_ioctl.o
  CC      drivers/char/hello.o
  CC      drivers/char/pty.o
  CC      drivers/char/misc.o
  CC      drivers/char/vt_ioctl.o
  CC      drivers/char/vc_screen.o
  CC      drivers/char/consolemap.o
```

```
root@ubuntu:yulong_linux-2.6.9-eeliod# cd arch/arm/boot/
root@ubuntu:boot# ls
bootp  compressed  Image  install.sh  Makefile  zImage
root@ubuntu:boot#
```

四、内核模块的编译

1、找到内核模块编译工具所在位置，并在 makefile 中赋值给 KDIR

```
root@ubuntu:yulong# cd /
root@ubuntu:/# cd usr/src/
root@ubuntu:src# ls
linux-headers-2.6.35-25  linux-headers-2.6.35-25-generic
root@ubuntu:src# cd linux-headers-2.6.35-25-generic/
root@ubuntu:linux-headers-2.6.35-25-generic# ls
arch    Documentation  fs       ipc     lib       Module.
block   drivers        include  Kbuild  Makefile  net
crypto  firmware       init     kernel  mm        samples
root@ubuntu:linux-headers-2.6.35-25-generic# pwd
/usr/src/linux-headers-2.6.35-25-generic
root@ubuntu:linux-headers-2.6.35-25-generic#
```

2、编辑模块文件 helloworld.c 和编译文件 Makefile。

本部分使用老师写好的代码，不再赘述。

此部分注意事项：由于内核是模块化形式，因此不能直接使用 gcc 进行编译，而将其写入 Makefile 文件中。

3、使用 make 命令执行，对 helloworld.c 文件进行编译

```
root@ubuntu:桌面# cd yulong
root@ubuntu:yulong# make
make -C /usr/src/linux-headers-2.6.35-25-generic M=/r
make[1]: 正在进入目录 `/usr/src/linux-headers-2.6.35-
  CC [M]  /root/桌面/yulong/helloworld.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/桌面/yulong/helloworld.mod.o
  LD [M]  /root/桌面/yulong/helloworld.ko
make[1]:正在离开目录 `/usr/src/linux-headers-2.6.35-2
```

4、使用 insmod 对生成的编译文件 helloworld.ko 文件进行加载，并使用 lsmod 查看是否加载成功

```
root@ubuntu:yulong# insmod helloworld.ko
root@ubuntu:yulong# lsmod
Module                  Size  Used by
helloworld               605  0
vmhgfs                 44018  0
```

5、通过 dmesg 指令查看模块输出信息

```
[    3.413163] NET: Registered pro
[    5.840757] EXT4-fs (sda1): re-
[   13.393944] eth0: no IPv6 route
[  104.307695] Hello world!
root@ubuntu:yulong#
```

6、使用 rmmod 指令进行模块卸载，并查看输出信息

```
[  104.307695] Hello world!
[  184.316865] Goodbye world!
root@ubuntu:yulong#
```