

# 嵌入式Linux操作系统作业

---

## I. 安装Linux操作系统

---

发行版版本号

Ubuntu 16.04.2 LTS

Linux内核版本号

Linux-4.8.0-58-generic

版本特点

- 支持U盘启动安装，安装过程简单流畅
- 接近Windows系统的图形化界面，比较方便日常使用
- 命令行终端操作灵活方便
- 网络安全性高于Windows系统，wireshark抓不到包
- ... ..

## II. QEMU中运行文件系统和内核

---

实验内容（除去QEMU和内核）均在所附tiny\_zhz文件中。本次试验需要使用qemu模拟器来模拟一个安装了linux的CPU，模拟器需要加载linux内核镜像和其相关的文件系统镜像。实验步骤如下。

安装QEMU

执行" sudo apt-get install qemu "安装qemu，完成后进行测试——输入" qemu "后双击Tab键，列出QEMU目前支持的所有CPU类型如下。

```
computer@computer-Lenovo-Product-Invalid-entry-length-16-Fixed-up-to-11: ~  
computer@computer-Lenovo-Product-Invalid-entry-length-16-Fixed-up-to-11:~$ qemu-  
qemu-aarch64          qemu-ppc64          qemu-system-mipsel  
qemu-alpha            qemu-ppc64abi32     qemu-system-moxie  
qemu-arm              qemu-ppc64le        qemu-system-or32  
qemu-armeb            qemu-s390x          qemu-system-ppc  
qemu-cris              qemu-sh4             qemu-system-ppc64  
qemu-i386              qemu-sh4eb          qemu-system-ppc64le  
qemu-img              qemu-sparc           qemu-system-ppcemb  
qemu-io               qemu-sparc32plus     qemu-system-sh4  
qemu-m68k              qemu-sparc64         qemu-system-sh4eb  
qemu-make-debian-root qemu-system-aarch64  qemu-system-sparc  
qemu-microblaze        qemu-system-alpha    qemu-system-sparc64  
qemu-microblazeel      qemu-system-arm      qemu-system-tricore  
qemu-mips              qemu-system-cris     qemu-system-unicore32  
qemu-mips64            qemu-system-i386     qemu-system-x86_64  
qemu-mips64el          qemu-system-lm32     qemu-system-xtensa  
qemu-mipsel            qemu-system-m68k     qemu-system-xtensaeb  
qemu-mipsn32           qemu-system-microblaze qemu-tilegx  
qemu-mipsn32el         qemu-system-microblazeel  
qemu-nbd               qemu-system-mips     qemu-unicore32  
qemu-or32              qemu-system-mips64   qemu-x86_64  
qemu-ppc               qemu-system-mips64el  
computer@computer-Lenovo-Product-Invalid-entry-length-16-Fixed-up-to-11:~$ qemu-
```

## Linux4.8内核镜像

执行如下命令，创建新的linux-4.8目录，用来存放linux内核源代码。

```
mkdir linux-4.8 # 创建内核源码目录
```

执行如下命令，创建新的obj文件夹，用来存放实验运行的环境（内核镜像+文件系统镜像）。

```
mkdir obj # 创建实验环境目录
```

在linux-4.8目录下，执行如下命令，从网站下载linux源代码压缩包并解压到当前目录下。

```
curl https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.8.tar.xz | tar xJf - # 下载内核源码
```

执行如下命令，生成x86\_64内核的默认配置文件，并将其存放在obj/linux目录下，这样可以使得源码（linux-4.8目录下）与输出（obj/linux目录下）分离。

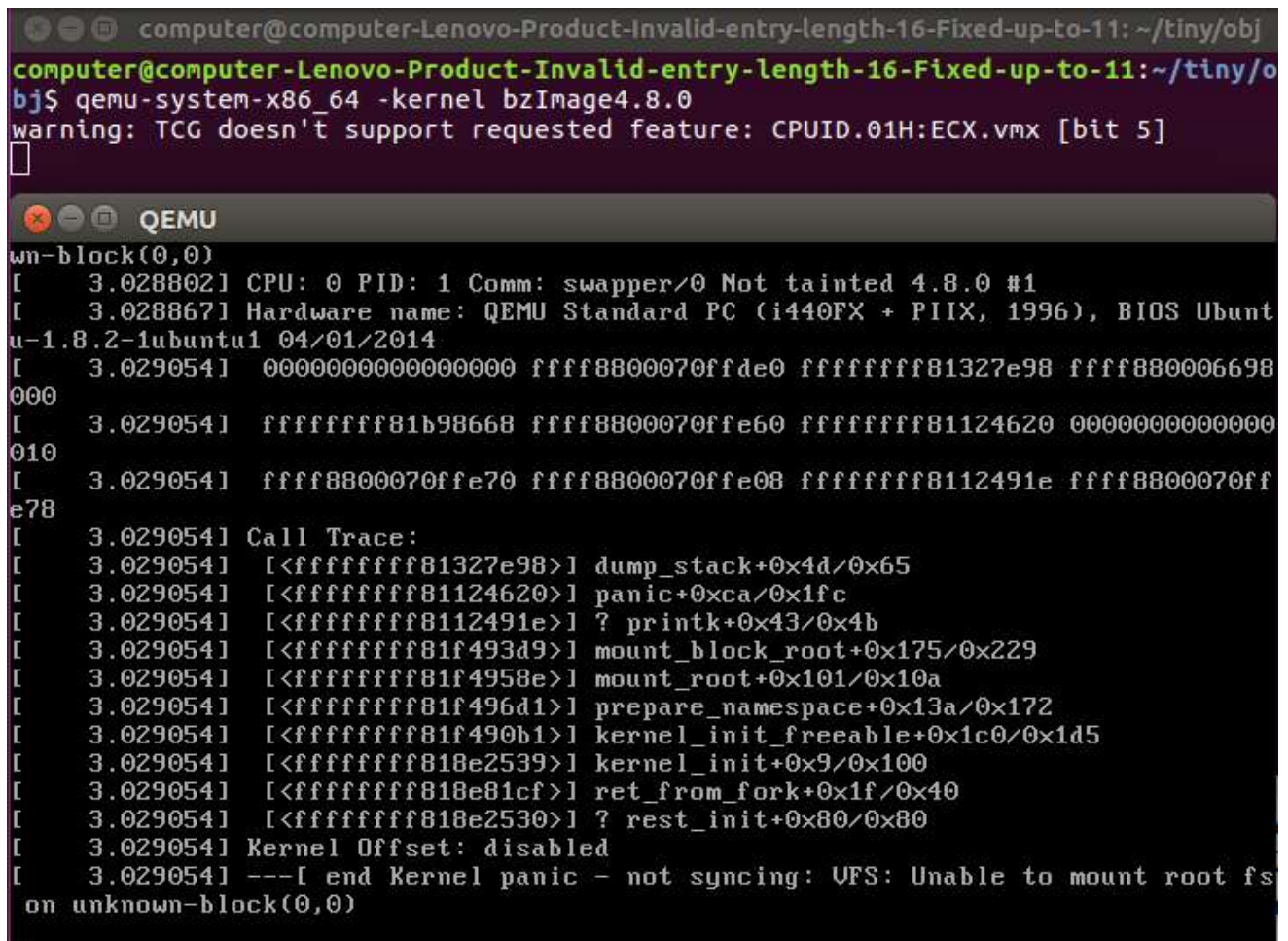
```
make O=../obj/linux x86_64_defconfig # 生成配置文件于另一目录下
```

切换至obj/linux目录，执行如下第一行命令，打开一个图形化界面，对内核进行配置，不进行更改直接退出即可。之后执行如下第二行命令，对内核进行编译，16线程并行可以加速编译。编译成功后得到输出如下，说明实验所需的内核镜像即在该路径下。

```
make O=../obj/linux x86_64_defconfig # 配置内核  
make -j16 # 16线程编译内核  
... ..  
Kernel: arch/x86/boot/bzImage is ready (#1)
```

将内核镜像bzImage拷贝至obj目录下，改名为bzImage4.8.0，之后使用qemu进行测试，命令如下。内核加载成功，但由于没有文件系统，内核崩溃，这说明目前为止的实验是成功的。

```
qemu-system-x86_64 -kernel bzImage4.8.0 # 测试加载内核
```



## Busybox制作文件系统镜像

执行如下命令，创建新的busybox-1.26.0目录，用来存放Busybox源码。

```
mkdir busybox-1.26.0 # 创建busybox源码目录
```

在busybox-1.26.0文件夹下，执行如下第一条命令，生成默认配置文件，并将其存放在obj/busybox目录下。这么做的原因与之前内核编译时相同。

```
make O=../obj/busybox defconfig # 生成配置文件
cd ../obj/busybox                # 切换至配置文件所在目录
make menuconfig                  # 配置busybox
```

在配置busybox时，需要修改配置，使用静态链接，修改方法如下。否则程序运行时需要加载动态库，那么就必须要将动态库拷贝到文件系统中。

```
-> Busybox Settings
-> Build Options
  [*] Build BusyBox as a static binary (no shared libs)
```

在obj/busybox目录下，执行如下命令进行编译，并且生成\_install目录

```
make          # 编译busybox
make install  # 生成_install目录
```

在obj目录下，执行如下命令，创建新的ramdisk目录，用来存放整个文件系统，最终的文件系统镜像即为将该目录打包生成的。

```
mkdir ramdisk # 创建文件系统目录
```

在ramdisk目录下，执行如下第一行的命令，将busybox/\_install目录下所有内容，拷贝到ramdisk目录中。执行如下第二行的命令，补充上文件系统需要的其他目录。

```
cp -r ../busybox/_install/* ./          # 拷贝文件系统目录
mkdir -pv {bin,sbin,mnt,usr,etc,proc,sys,dev,usr/bin,usr/sbin} # 补充文件系统目录
```

执行如下命令，将文件系统目录制作为cpio镜像文件，输出在obj目录下

```
find . | cpio --quiet -H newc -o | gzip -9 -n > ../initrd.gz
```

在obj目录下，使用qemu进行测试，命令如下。内核加载成功，但会与没有添加系统文件时一样，出现内核崩溃，说明系统文件制作有错误或者疏漏。

```
qemu-system-x86_64 -kernel bzImage4.8.0 -append "root=dev/ram0" -initrd initrd.gz
```

## 补充文件系统

内核崩溃的原因，是因为文件系统中没有init文件，因而内核无法启动init进程。之后的工作就是添加init文件，修复文件系统

在ramdisk/etc目录下执行如下命令，添加需要的init文件。

```
mkdir init.d
gedit inittab
... ..
gedit init.d/rcS
... ..
gedit mdev.config
```

inittab文件的内容如下

```

::sysinit:/etc/init.d/rcS      # init
... ..                        # 挂载设备等
::restart/sbin/init           # restart
::ctrlaltdel:/sbin/reboot     # ctrl-alt-del
::shutdown:/bin/umount -a -r  # shutdown

```

init.d/rcS文件的内容如下

```

#!/bin/sh
mount -t proc proc /proc
mount -t sysfs sysfs /sys
mdev -s

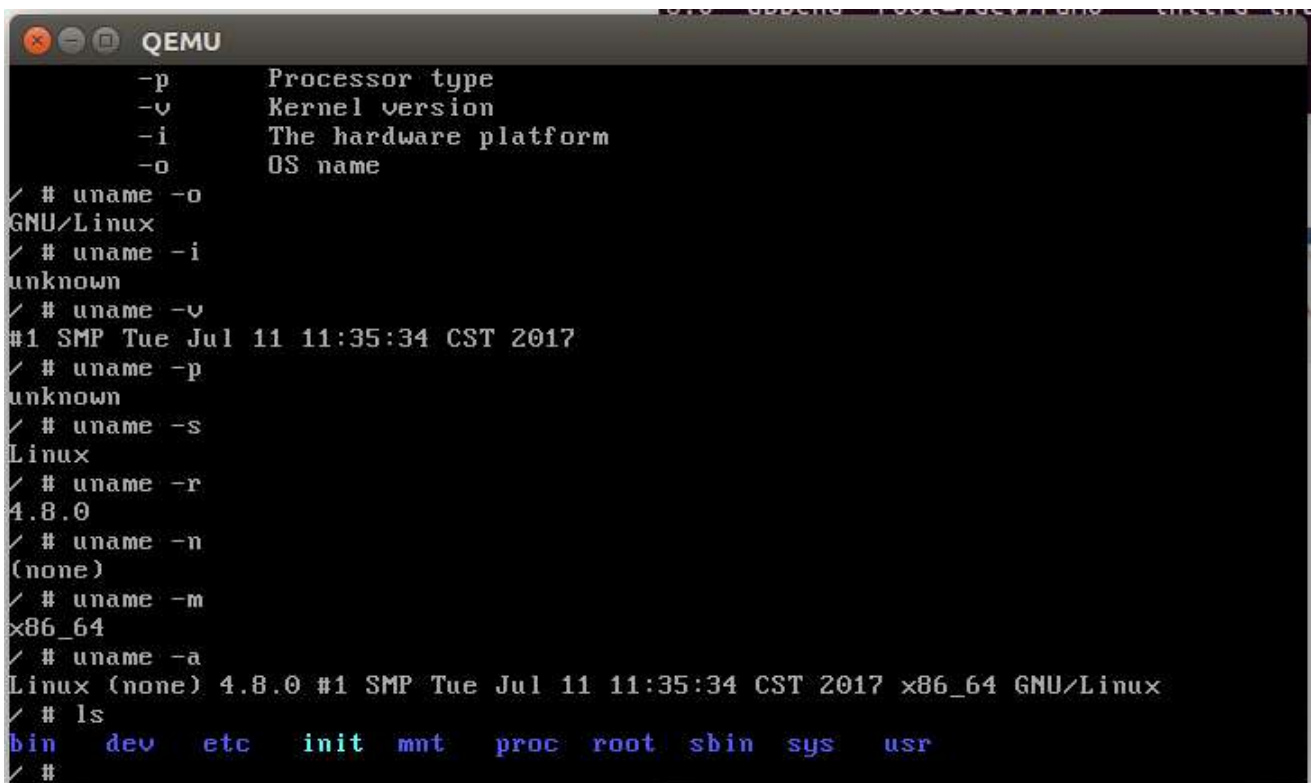
```

mdev.config文件中没有内容，为了防止mdev报错而存放的空的配置文件。之后在ramdisk目录下，再次执行如下命令，将文件系统目录制作为cpio镜像文件，输出在obj目录下。

```
find . | cpio --quiet -H newc -o | gzip -9 -n > ../initrd.gz # 生成文件系统镜像
```

在obj目录下，使用qemu进行测试，命令如下。内核加载成功，文件系统加载成功，虚拟机顺利运行。在qemu虚拟机中执行 "uname" 查看内核信息，如下图。

```
qemu-system-x86_64 -kernel bzImage4.8.0 -append "root=dev/ram0" -initrd initrd.gz #测试文件系统
```



```

QEMU
-p      Processor type
-v      Kernel version
-i      The hardware platform
-o      OS name
/ # uname -o
GNU/Linux
/ # uname -i
unknown
/ # uname -v
#1 SMP Tue Jul 11 11:35:34 CST 2017
/ # uname -p
unknown
/ # uname -s
Linux
/ # uname -r
4.8.0
/ # uname -n
(none)
/ # uname -m
x86_64
/ # uname -a
Linux (none) 4.8.0 #1 SMP Tue Jul 11 11:35:34 CST 2017 x86_64 GNU/Linux
/ # ls
bin  dev  etc  init  mnt  proc  root  sbin  sys  usr
/ #

```

### III. 设备驱动

实验内容（除去QEMU和内核）均在所附tiny\_zhz文件中。本次试验使用qemu模拟器挂载EDU教学设备，之后加载编写的EDU设备驱动读取其设备号、版本号等。实验步骤如下。

## Hello World驱动

在hello\_driver文件夹中编写最简单的Hello World驱动程序，C代码请见hello\_driver.c文件中，该程序不对应任何设备，仅仅在驱动模块加载和卸载时进行打印。

完成C程序编写后，编写Makefile如下。make后得到hello\_driver.o，将其移入QEMU虚拟机文件系统中，重新制作cpio镜像文件。

```
obj-m := hello_driver.o
KDIR := ../linux-4.8
PWD := $(shell pwd)
all:
    make -C $(KDIR) M=$(PWD) modules
clean:
    rm -f *.ko *.o *.mod.o *.mod.c *.symvers
```

hello\_driver.ko驱动位于文件系统/usr目录中。打开挂载了EDU设备的QEMU模拟器后，使用如下命令加载和卸载hello\_driver.ko模块，正常打印。

```
qemu-system-x86_64 -kernel bzImage4.8.0 -append "root=dev/ram0" -initrd initrd.gz -device edu #
挂载EDU设备测试驱动程序
#下面进入QEMU模拟器中
cd usr
insmod hello_world.ko      # 加载驱动模块，打印“Hello, beautiful world!”
lsmod                     # 显示已经加载的模块，hello_world.ko已经加载
rmmod hello_world.ko      # 卸载驱动模块，打印“Goodbye, cruel world!”
```

## EDU驱动

在edu\_driver文件夹中编写EDU设备的驱动程序，用于读取EDU设备的设备号、版本号等信息。

首先需要搜索驱动对应的设备，由于EDU设备是挂载在PCI上的设备，因此可以使用<linux/pci.h>提供的for\_each\_pci\_dev()函数遍历所有PCI设备，在其中寻找特定的Vendor ID和设备ID的设备（在本次实验中，EDU设备的Vendor ID为0x1234，Device ID为0x11E8）。

另一种方法是直接使用pci\_get\_device()函数寻找特定Vendor ID和设备ID的PCI设备，该函数返回一个struct pci\_dev类型的指针。

struct pci\_dev类型是PCI设备的描述符，其数据结构大致定义如下。

```
struct pci_dev
{
    ...
    unsigned short vendor;          // PCI设备的厂商ID [0:0]
    unsigned short device;         // PCI设备的设备ID [2:2]
    unsigned short subsystem_vendor; // PCI设备的子系统厂商ID [1:1]
    unsigned short subsystem_device; // PCI设备的子系统设备ID [3:3]
    ...
    struct pci_driver *driver;      // PCI设备的驱动程序指针
    ...
}
```



PCI设备在编程可见的空间中中存放的信息及其大小如下图所示。

Device ID [3:2]	Vendor ID [1:0]
Status [7:6]	Command [5:4]
Class [11:9]	Revision [8:8]
BIST [15:15]	Header Type [14:14]
	Latency Timer [13:13]
	Cache Line Size [12:12]
	Base Addr0 [19:16]
	Base Addr1 [23:20]
	Base Addr2 [27:24]
	Base Addr3 [31:28]
	Base Addr4 [35:32]
	Base Addr5 [39:36]
	Card CIS Pointer [43:40]
Subsystem Device [47:46]	Subsystem Vendor [45:44]

驱动模块加载后首先打印驱动和作者信息，之后按照如上所述的方式打印EDU设备各个信息如下图。

```
computer@computer-Lenovo-Product-Invalid-entry-length-16-Fixed-up-to-11:~/Desktop/tiny_zhz/obj$ qemu-system-x86_64 -kernel bzImage4.8.0 -append "root=/dev/ram0" -initrd initrd.gz -device edu
warning: TCG doesn't support requested feature: CPUID.01H:ECX.vmx [bit 5]

ls
bin          edu_driver.ko  sbin
/usr # insmod edu_driver.ko
[ 18.791342] edu_driver: loading out-of-tree module taints kernel.
[ 18.795404] Module initialized!
[ 18.795461] This EDU-device driver is written by
[ 18.795551]   Name: Zhang Huangzhao
[ 18.795596]   PKU ID: 1400017707
[ 18.795644] Reading PCI EDU-Device 1234:11e8
[ 18.795737] PCI configuration space:
[ 18.795824]   [3:0] = 11e81234
[ 18.795876]   Vendor[1:0] = 1234; Device[3:2] = 11e8
[ 18.795938]   [7:4] = 00000103
[ 18.795989]   Command[5:4] = 0103; Status[7:6] = 0000
[ 18.796075]   [11:8] = 00ff0010
[ 18.796208]   Revision[8:8] = 10; Class[11:9] = 00ff00
[ 18.796280]   [15:12] = 00000000
[ 18.796337]   Cache Line Size[12:12] = 00; Latency Timer[13:13] = 00; Header Type[14:14] = 00; BIST[15:15] = 00
[ 18.796430]   [47:44] = 11001af4
[ 18.796482]   Subsystem Vendor[45:44] = 1af4; Subsystem ID[47:46] = 1100
/usr # _
```

# IV. Qt小游戏

安装Qt环境

从Qt官网下载在线安装程序包，安装Qt5.8版本。

安装完成后，需要在安装目录下才能启动Qt Creator进行使用。为了可以直接在任意目录下使用命令行开启，按照官方手册编写脚本如下。

```
[Desktop Entry]
Encoding=UTF-8
Name=QtCreator
Comment=QtCreator
Exec=/home/lc/QT/Tools/QtCreator/bin/qtcreator
Icon=/home/lc/QT/Tools/QtCreator/share/qtcreator/templates/shared/icon64/icon64.png
Terminal=false
StartupNotify=true
Type=Application
Categories=Application;Development;
```

将其放置在~/.local/share/applications目录下，之后编辑在同一目录下的defaults.list文件，添加如下的一行。

```
text/qtcreator=Qt-Creator.desktop;
```

最后运行如下命令更新配置，更新完成后可以在命令行直接输入"qtcreator"打开Qt Creator。

```
sudo update-mime-database /usr/share/mime
```

## 编写QtRoshambo小游戏

roshambo.h和roshambo.cpp文件声明并实现了Roshambo类，该类中包含石头剪刀布游戏的执行逻辑。

mainwindow.ui文件中存储了设计好的图形界面格式，会加载入MainWindow类中。

mainwindow.h和mainwindow.cpp文件加载游戏GUI并将游戏逻辑接入，使用了Qt提供的多种控件和多媒体组件，也对Windows环境下的应用图标进行了设置。

main.cpp文件为游戏的入口，创建一个MainWindow对象并运行。

bmp/figures/icon.bmp为Windows环境下的程序图标的源bmp格式文件。

app.rc文件为设置游戏图标的资源文件。

## 运行QtRoshambo

在Windows环境下编译源码或直接运行Release目录下的QtRoshambo.exe。

游戏运行过程中的部分截图如下。



