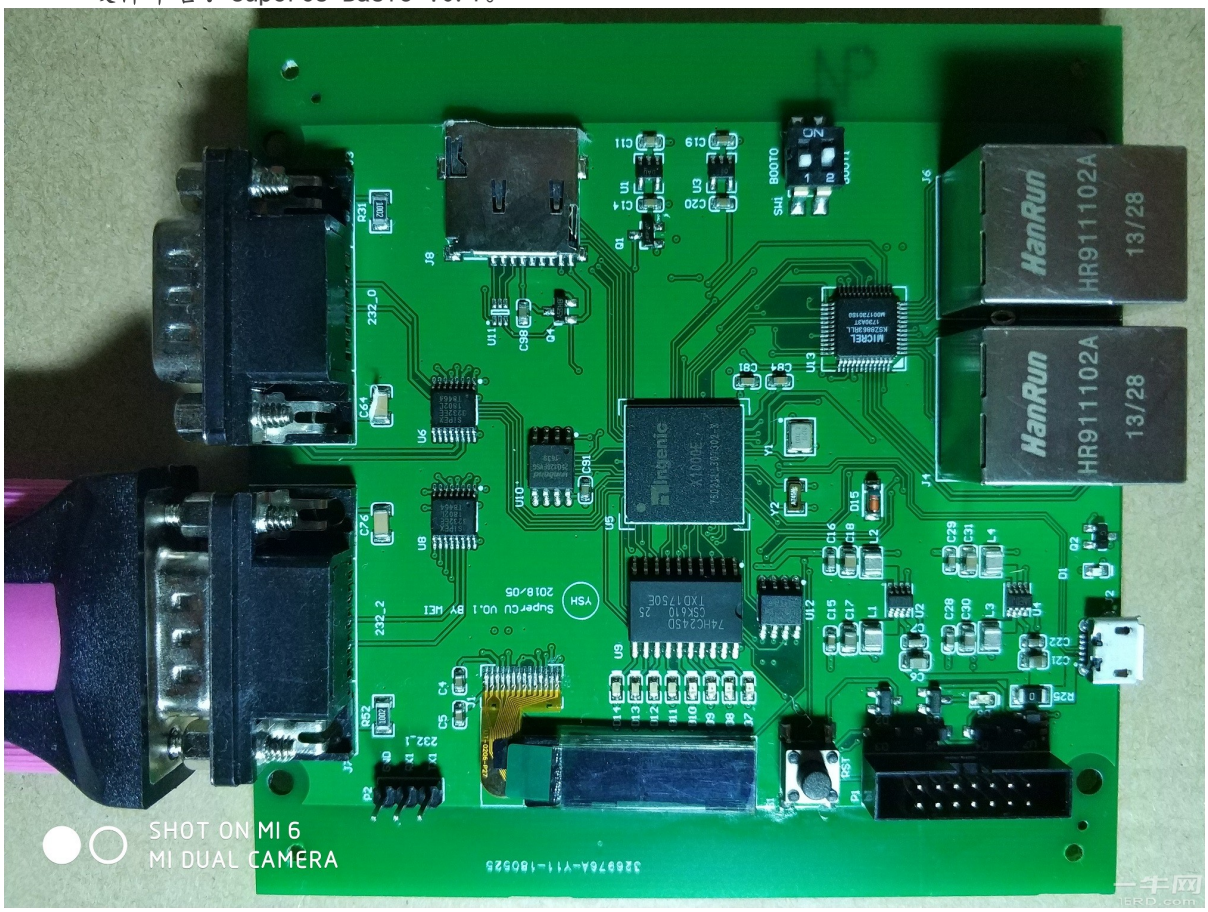
在树莓派和基于君正JZ4780的CI20板子上都跑过Debian8，突发奇想是否能把Debian8裁剪之后在君正X1000E这一类的轻量化平台上跑起来，先开个帖子，近期准备来折腾折腾。

**说一说大致的思路，预计有以下几个步骤：**

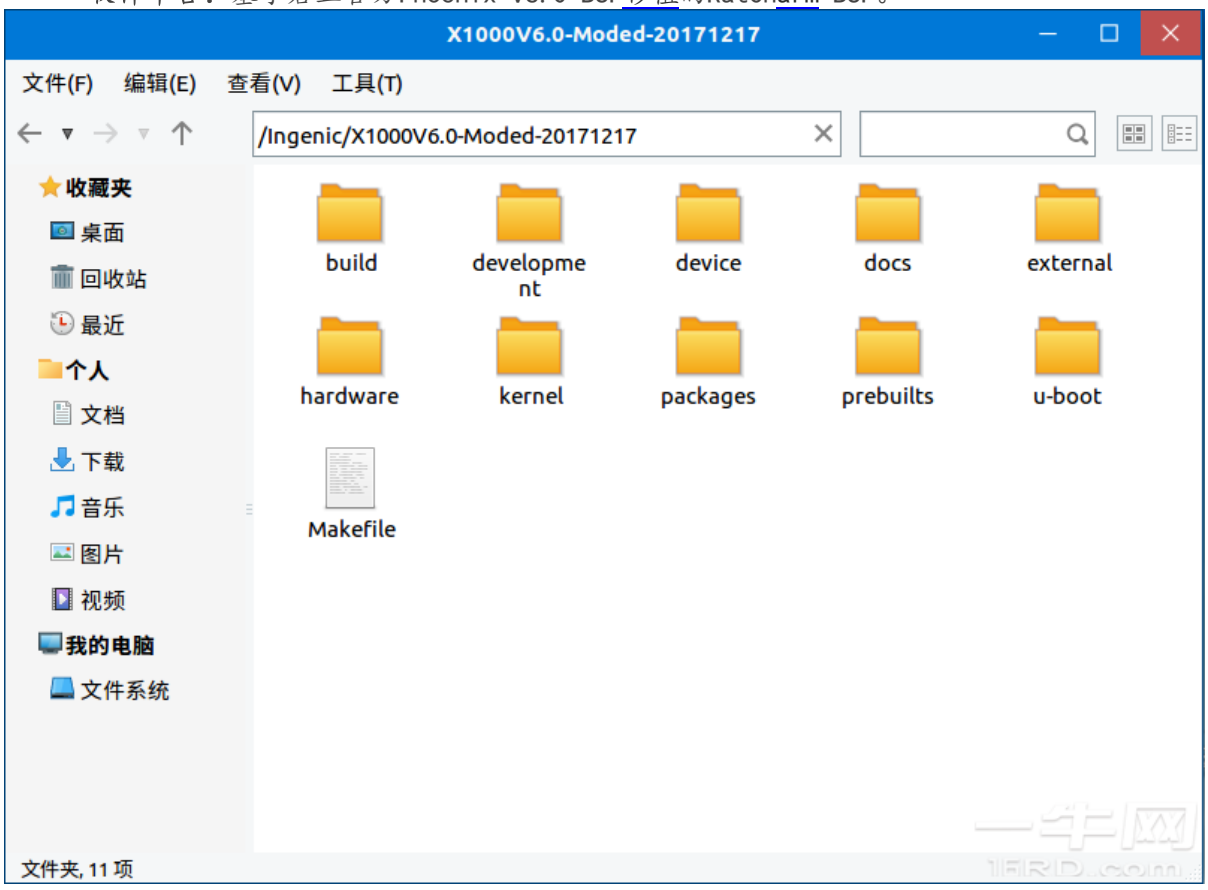
1. 基于已有的BSP，实现由TF卡启动系统。
2. 打包Debian8的最小rootfs，替代原BSP中的Buildroot。
3. 解决各种出现的bug。

**基于现有BSP已经成功通过TF卡启动，下一步将尝试引导Debian8的Rootfs，基本情况如下：**

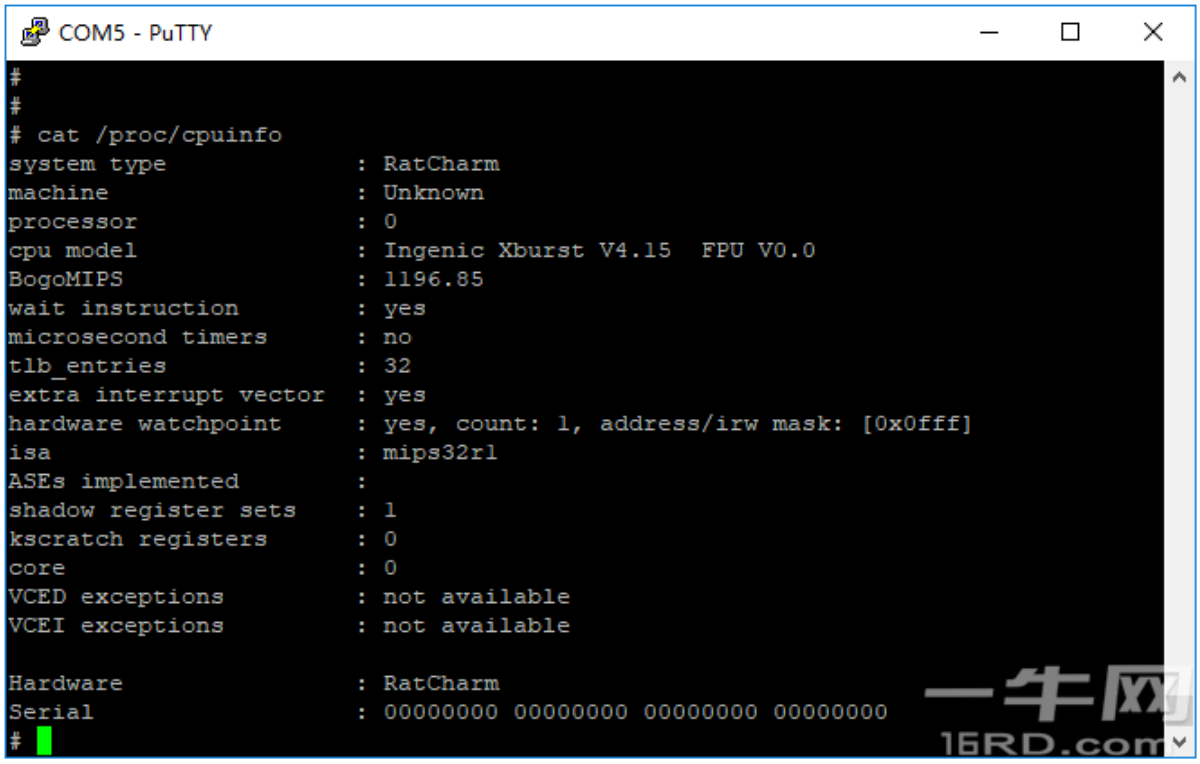
- 硬件平台：SuperCU Basic V0.1。




- 软件平台：基于君正官方Phoenix v6.0 BSP移植的RatCharm BSP。



- 成果：已成功通过TF卡启动Build Root系统，并通过命令行查看/proc/cpuinfo信息。




- 以下是移植步骤：

1. 修改并编译用于TF卡启动的uboot。
   因为所用硬件平台引出的调试串口是复用于Port D的UART2，因此需要修改打印串口的配置。
   将u-boot文件夹中include/configs/RatCharm.h中关于主频的配置进行修改，以下是原始代码：

```
#define CONFIG_SYS_APLL_FREQ              1008000000    /*If APLL not use mast be set 0*/
#define CONFIG_SYS_MPLL_FREQ              600000000     /*If MPLL not use mast be set 0*/
#define CONFIG_CPU_SEL_PLL                APLL
#define CONFIG_DDR_SEL_PLL                MPLL
#define CONFIG_SYS_CPU_FREQ               1008000000
#define CONFIG_SYS_MEM_FREQ               200000000
```

复制代码

修改后的代码如下：

```
#define CONFIG_SYS_APLL_FREQ              1200000000    /*If APLL not use mast be set 0*/
#define CONFIG_SYS_MPLL_FREQ              600000000     /*If MPLL not use mast be set 0*/
#define CONFIG_CPU_SEL_PLL                APLL
#define CONFIG_DDR_SEL_PLL                MPLL
#define CONFIG_SYS_CPU_FREQ               1200000000
#define CONFIG_SYS_MEM_FREQ               200000000
```

复制代码

随后对同一文件下的关于打印串口的配置部分进行修改，以下是原始代码：

```
#define CONFIG_SYS_UART_INDEX                 2
#if defined(CONFIG_RatCharm_BARE) || defined(CONFIG_RatCharm_RTTHREAD)
#define CONFIG_SYS_UART2_PD
#define CONFIG_BAUDRATE                       115200
#else
#define CONFIG_SYS_UART2_PC
#define CONFIG_BAUDRATE                       115200
#endif
```

复制代码

修改后的代码如下：

```
#define CONFIG_SYS_UART_INDEX                   2
#if defined(CONFIG_RatCharm_BARE) || defined(CONFIG_RatCharm_RTTHREAD)
#define CONFIG_SYS_UART2_PD
#define CONFIG_BAUDRATE                         115200
#else
#define CONFIG_SYS_UART2_PD
#define CONFIG_BAUDRATE                         115200
#endif
```

复制代码

修改完毕后使用以下命令进行编译，编译耗时大约1分钟左右，待编译完成后，
即可在u-boot目录下得到用于TF卡启动的u-boot-with-spl-mbr-gpt.bin文件。

```
make RatCharm_uImage_msc0 -j4
```

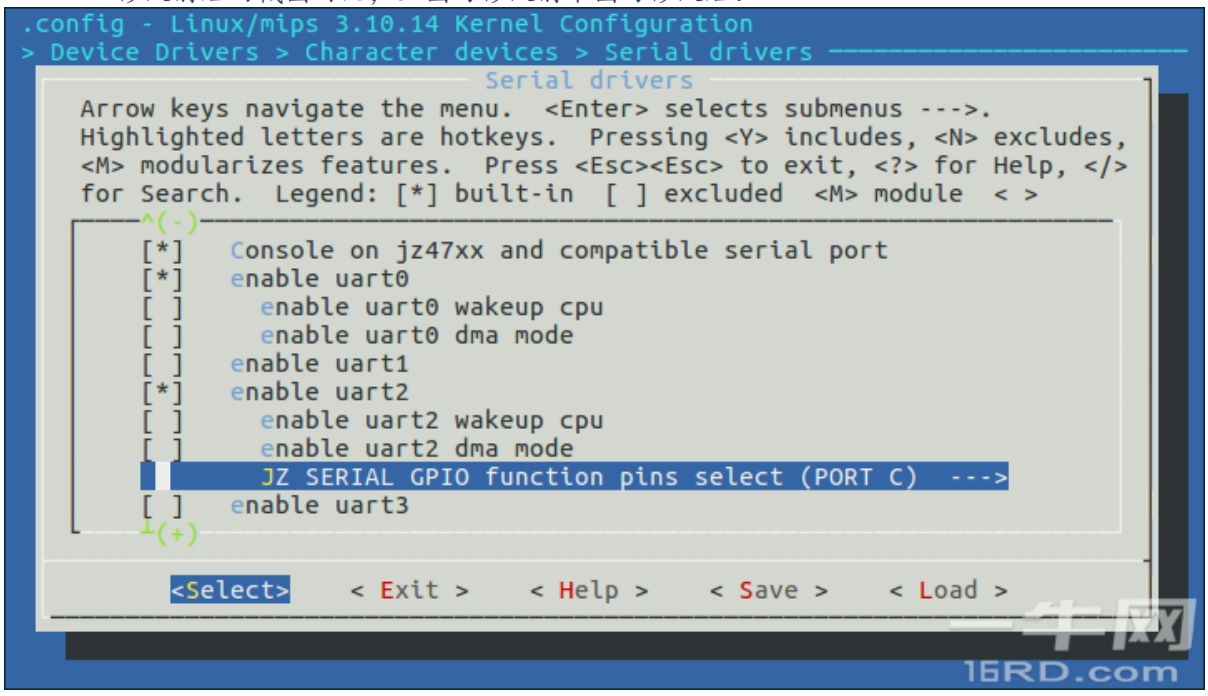
复制代码

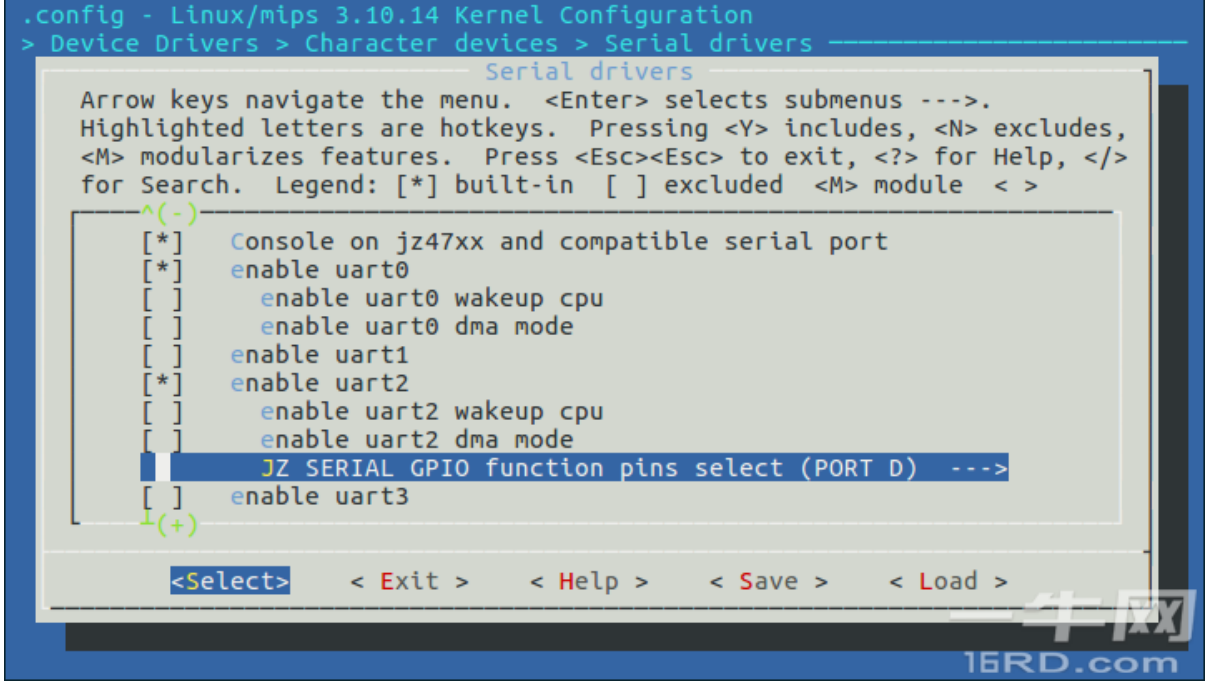
2. 修改并编译用于TF卡启动的kernel。（以make menuconfig方式为例）
前面提到过，由于调试串口使用的是Port D的UART2，因此首先需要在内核配置中修改相应配置。
首先进入到kernel目录，使用make RatCharm_linux_defconfig进行默认配置，
然后将原来默认的Port C端口修改为Port D端口，具体如下：

```
| Symbol: SERIAL_JZ47XX_UART2_PD [=y]                                          |
| Type  : boolean                                                              |
| Prompt: PORT D                                                               |
|   Location:                                                                  |
|     -> Device Drivers                                                        |
|       -> Character devices                                                   |
|         -> Serial drivers                                                    |
|           -> ingenic jz47xx serial port support (SERIAL_JZ47XX_UART [=y])    |
|             -> enable uart2 (SERIAL_JZ47XX_UART2 [=y])                       |
|               -> JZ SERIAL GPIO function pins select (<choice> [=y])        |
|   Defined at drivers/tty/serial/Kconfig:1575                                 |
|   Depends on: <choice> && SOC_X1000 [=y]                                     |
```

复制代码

修改前后的截图对比，上图为修改前下图为修改后：



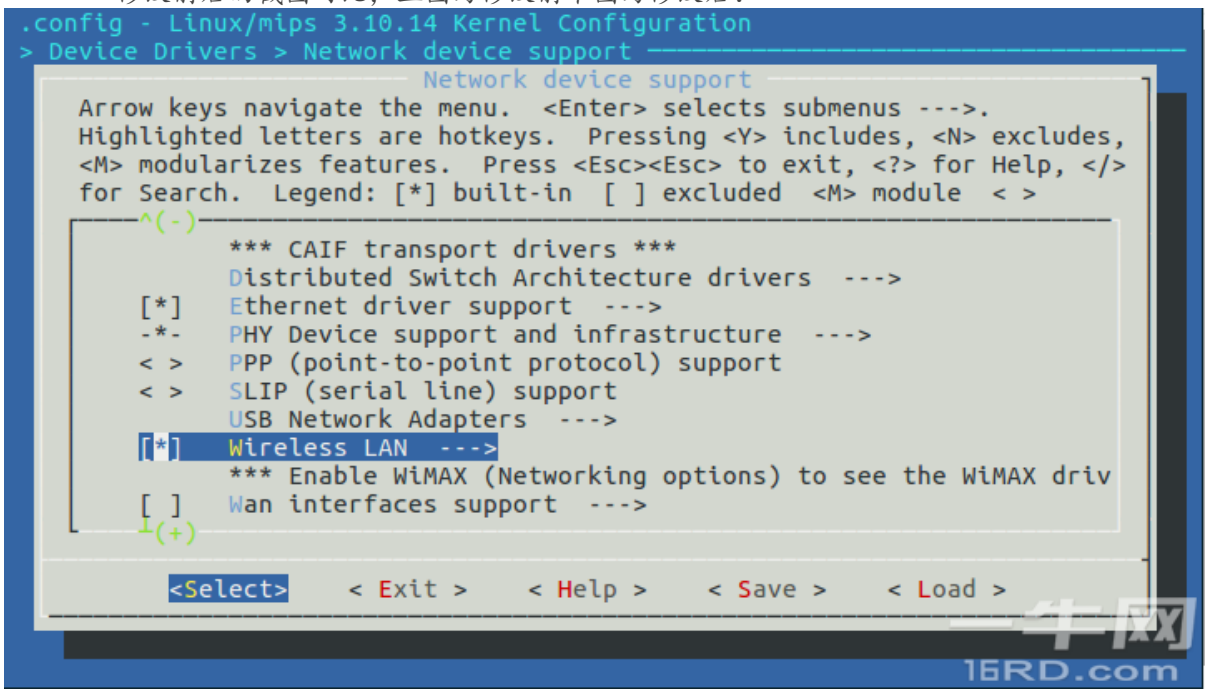


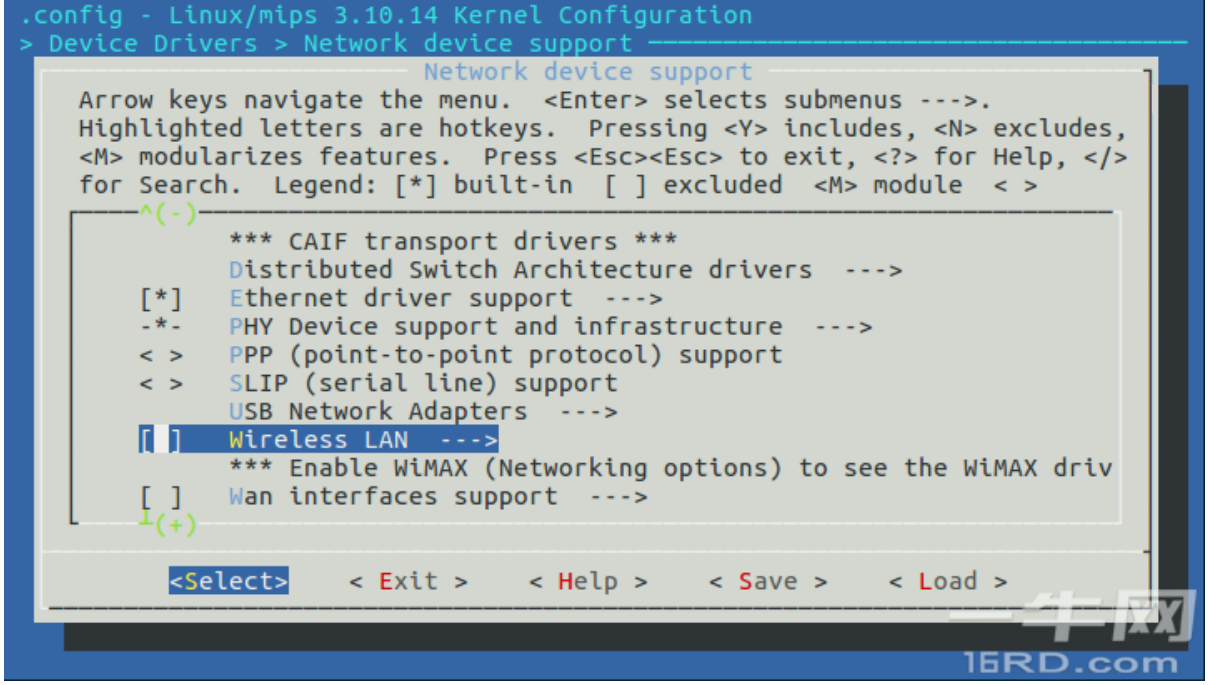

现在的平台没有无线通信模块，因此修改相应配置关闭无线网络和蓝牙功能，共需修改三处：
第一处，关闭Wireless LAN，具体如下：

```
01.     | Symbol: WLAN [=n]                                                    |
02.     | Type  : boolean                                                      |
03.     | Prompt: Wireless LAN                                                 |
04.     |   Location:                                                          |
05.     |     -> Device Drivers                                                |
06.     |       -> Network device support (NETDEVICES [=y])                    |
07.     |   Defined at drivers/net/wireless/Kconfig:5                          |
08.     |   Depends on: NETDEVICES [=y] && !S390 && NET [=y]                   |
09.     |   Selects: WIRELESS [=n]                                             |
```

复制代码

修改前后的截图对比，上图为修改前下图为修改后：



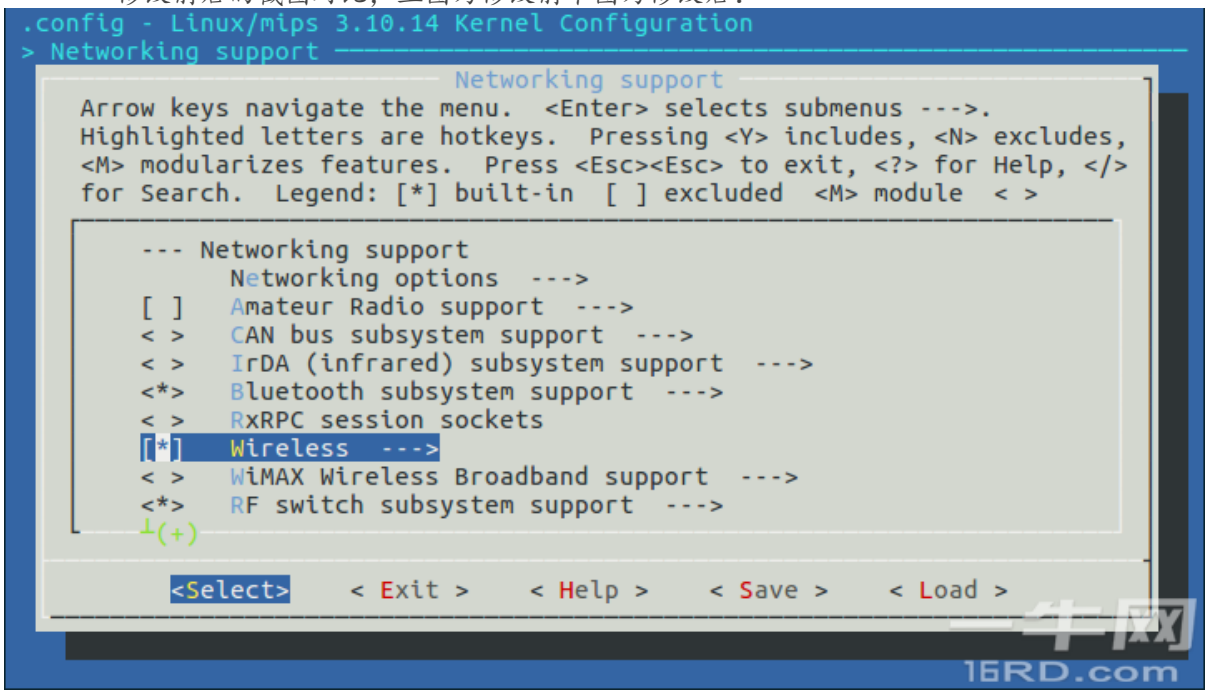


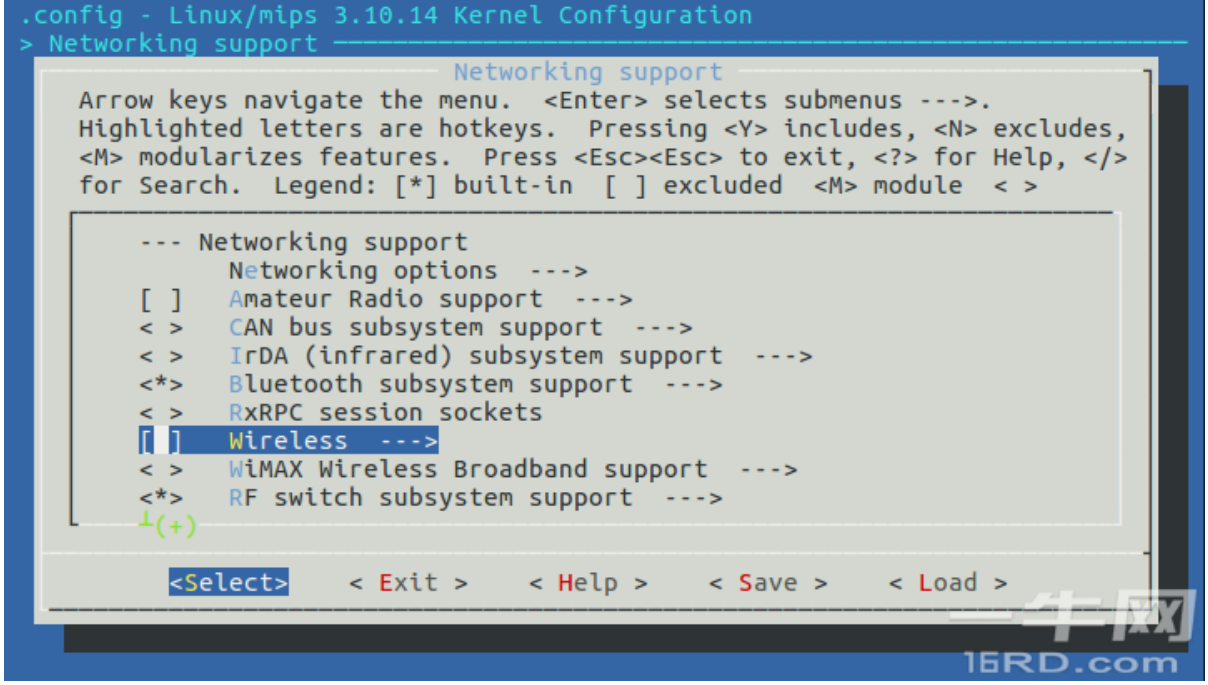

第二处，关闭Wireless，具体如下：

```
01.     | Symbol: WIRELESS [=n]                                                 |
02.     | Type  : boolean                                                       |
03.     | Prompt: Wireless                                                      |
04.     |   Location:                                                           |
05.     |     -> Networking support (NET [=y])                                  |
06.     |   Defined at net/Kconfig:334                                          |
07.     |   Depends on: NET [=y] && !S390                                       |
08.     |   Selected by: WLAN [=n] && NETDEVICES [=y] && !S390 && NET [=y]      |
```

复制代码

修改前后的截图对比，上图为修改前下图为修改后：



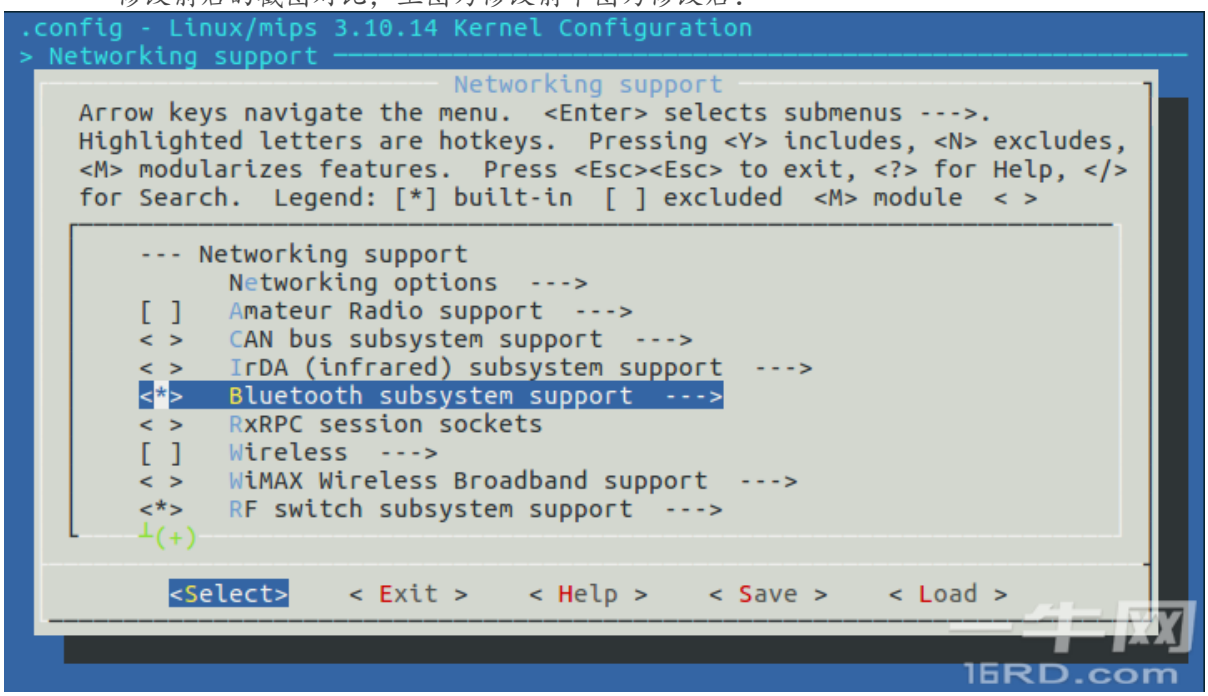


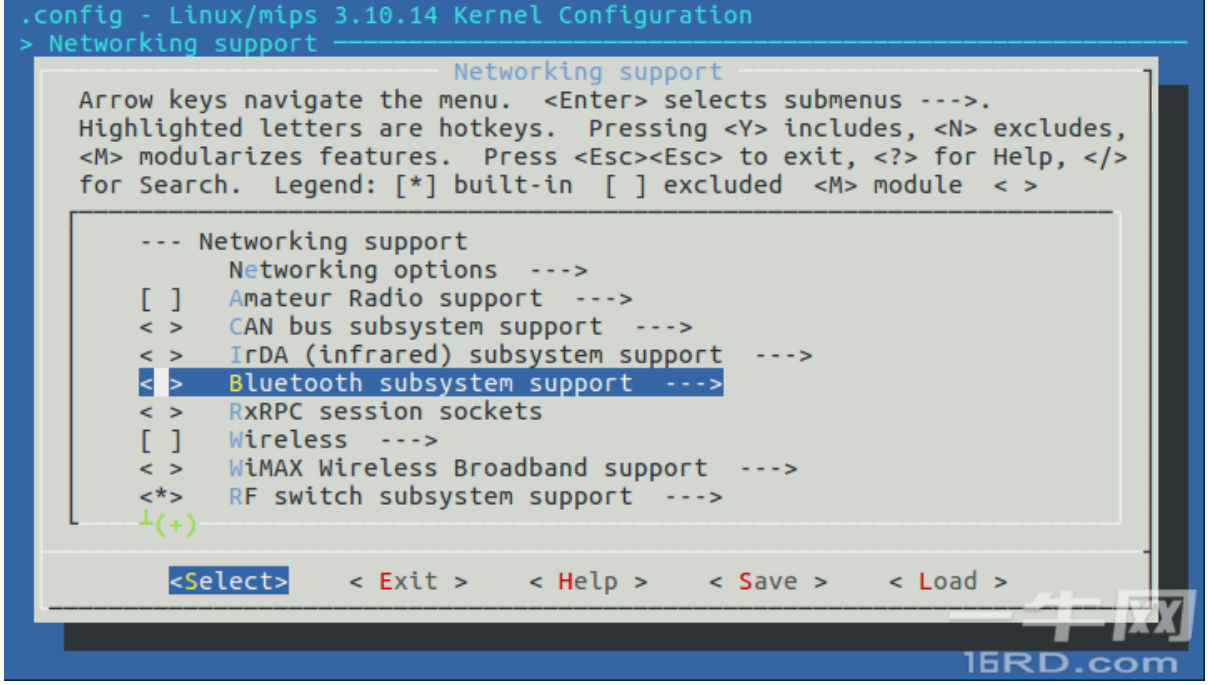

第三处，关闭Bluetooth subsystem support，具体如下：

```
01.     | Symbol: BT [=n]                                                       |
02.     | Type  : tristate                                                      |
03.     | Prompt: Bluetooth subsystem support                                   |
04.     |   Location:                                                           |
05.     |     -> Networking support (NET [=y])                                  |
06.     |   Defined at net/bluetooth/Kconfig:5                                  |
07.     |   Depends on: NET [=y] && !S390 && (RFKILL [=n] || !RFKILL [=n])      |
08.     |   Selects: CRC16 [=y] && CRYPTO [=y] && CRYPTO_BLKCIPHER [=y] && \    |
09.     | CRYPTO_AES [=y] && CRYPTO_ECB [=y] && CRYPTO_SHA256 [=y]              |
```

复制代码

修改前后的截图对比，上图为修改前下图为修改后：



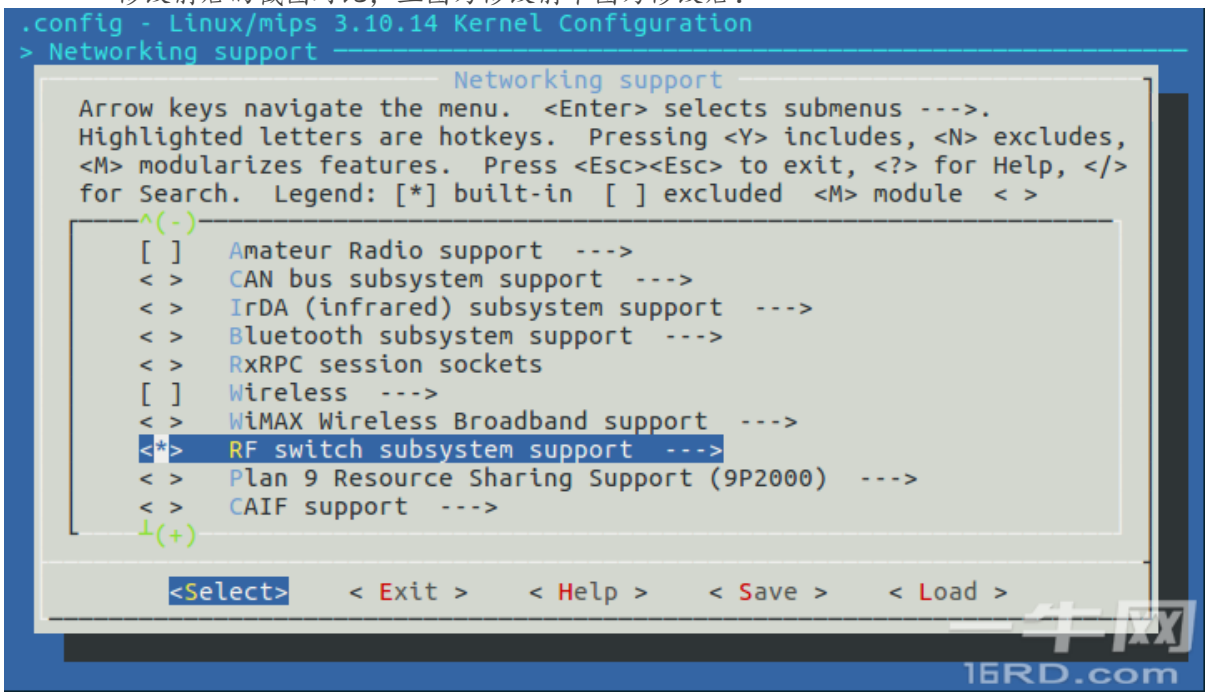


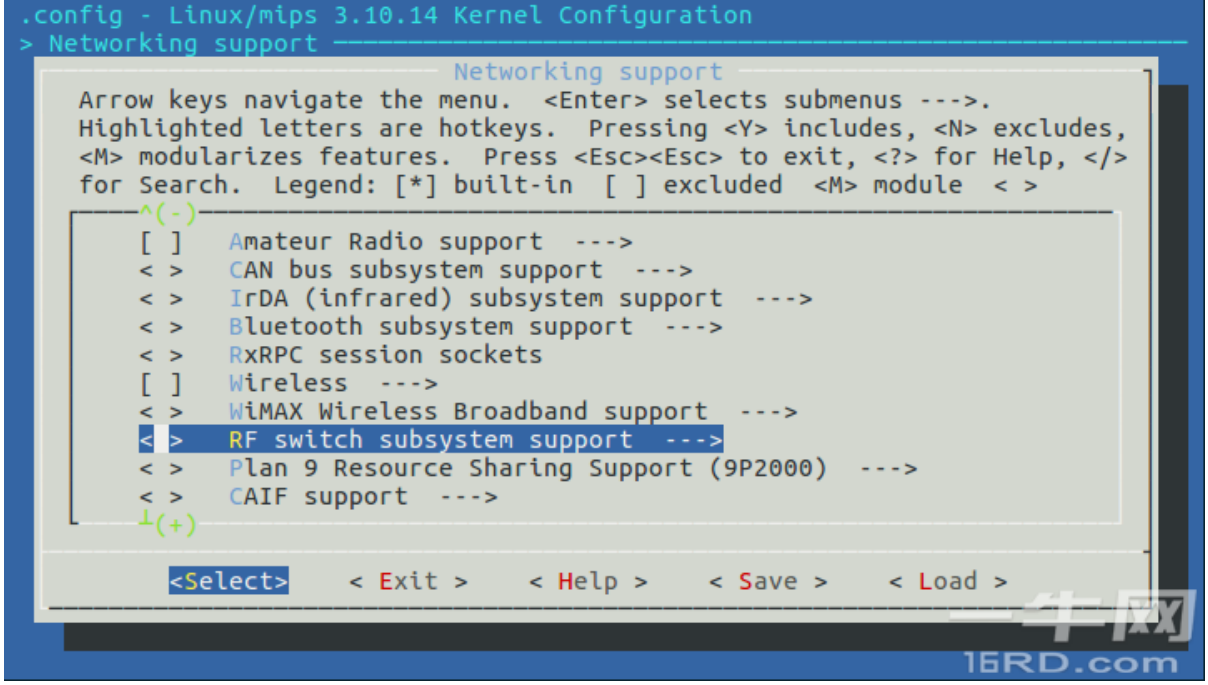

关闭无线网络及蓝牙相关的功能后，相应的功率管制功能也需要关闭，共需修改两处。
第一处，关闭RF switch subsystem support，具体如下：

```
01.     | Symbol: RFKILL [=n]                                                  |
02.     | Type  : tristate                                                     |
03.     | Prompt: RF switch subsystem support                                  |
04.     |   Location:                                                          |
05.     |     -> Networking support (NET [=y])                                 |
06.     |   Defined at net/rfkill/Kconfig:4                                    |
07.     |   Depends on: NET [=y]                                               |
```

复制代码

修改前后的截图对比，上图为修改前下图为修改后：



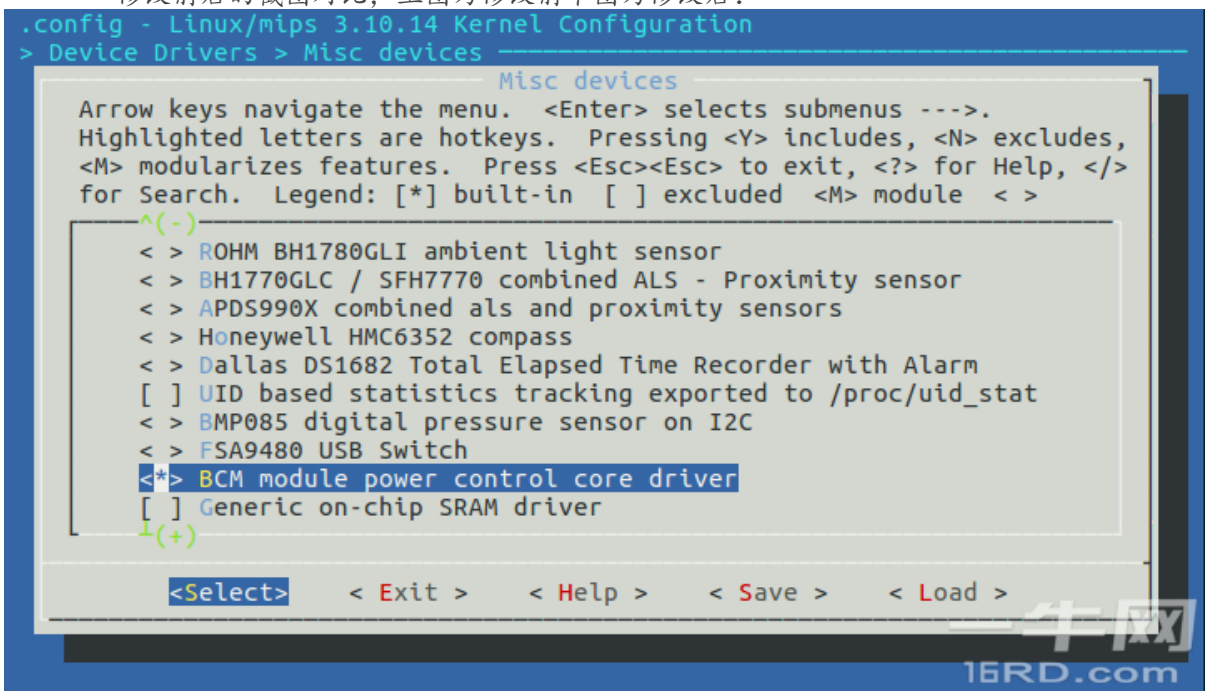


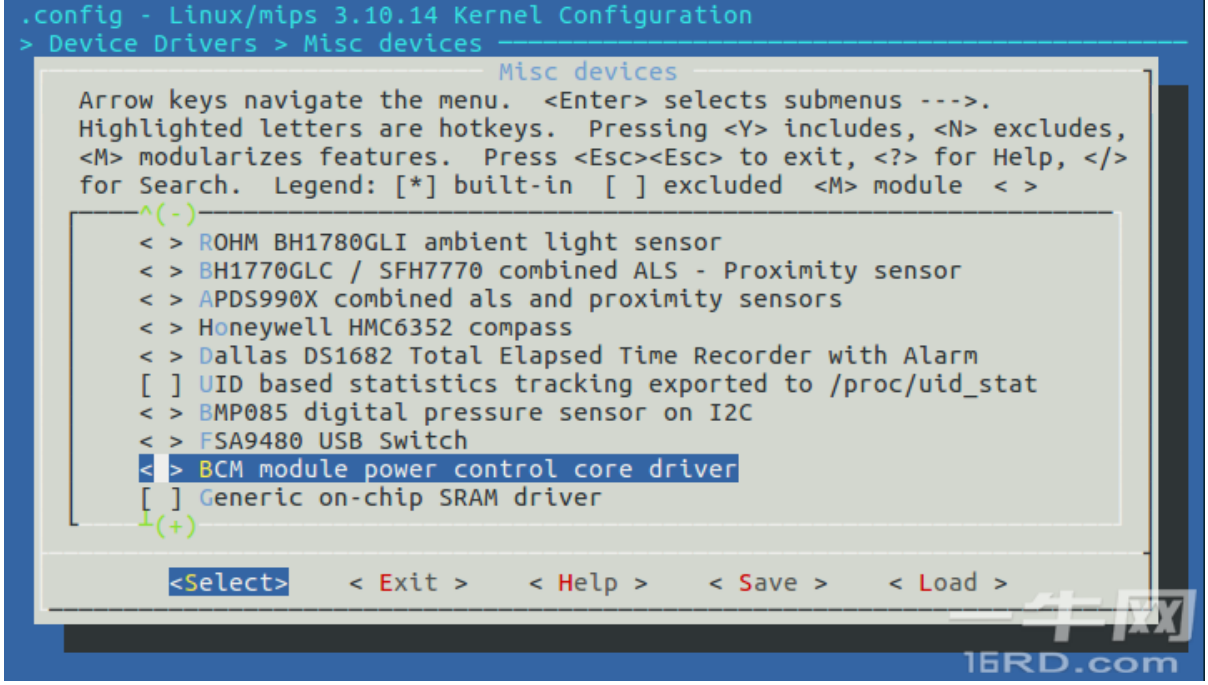

第二处，关闭BCM module power control core driver，具体如下：

```
01.     | Symbol: BCM_PM_CORE [=n]                                    |
02.     | Type  : tristate                                            |
03.     | Prompt: BCM module power control core driver                |
04.     |   Location:                                                 |
05.     |     -> Device Drivers                                       |
06.     |       -> Misc devices                                       |
07.     |   Defined at drivers/misc/Kconfig:545                       |
```

复制代码

修改前后的截图对比，上图为修改前下图为修改后：






完成所有修改后使用以下命令进行编译，编译耗时大约3至5分钟，待编译完成后，
即可在kernel/arch/mips/boot目录下得到用于TF卡启动的uImage文件。

```
make uImage -j4
```

复制代码

3. 打包ext4格式的Build Root文件系统。
由于是使用TF卡启动，因此文件系统需要制作成ext4格式，具体步骤如下：
（1）进入到BSP根目录，选择RatCharm_norflash-user对BSP进行一次完整编译。
（2）此时可在out/product/RatCharm路径下找到制作rootfs所需的system目录。
（3）进入以上路径（不要进入system目录）后依次执行以下命令：

```
dd if=/dev/zero of=system.ext4 bs=1M count=512
mkfs.ext4 system.ext4
sudo mount -o loop system.ext4 /mnt
sudo cp -a -fr system/* /mnt
sync
sudo umount /mnt
sudo chmod 777 system.ext4
```

复制代码

若顺利完成则可得到名为system.ext4的目标文件，
使用cloner烧录以上步骤得到的三个文件即可实现从TF卡启动。

- 附上RatCharm平台BSP包及已编译好的用于TF卡启动的目标文件：

RatCharm平台BSP包：

X1000V6.0-Moded-20171217.part1.rar （450 MB， 下载次数：8 ）

X1000V6.0-Moded-20171217.part2.rar （430.67 MB， 下载次数：7 ）

用于TF卡启动的目标文件：

u-boot-with-spl-mbr-gpt.zip （95.18 KB， 下载次数：2 ）

uImage.zip （2.4 MB， 下载次数：2 ）

system.zip （9.35 MB， 下载次数：2 ）

**接下来将尝试引导Debian8的Rootfs，预计会有以下几个步骤：**

1. 打包Debian8的最小rootfs并制作成ext4格式镜像。
2. 用Debian8的rootfs镜像替代此前的Buildroot。
3. 尝试引导Debian8并解决出现的各种问题。

- 以下是移植步骤：

1. 修改并使用mkdebianrfs.sh脚本打包Debian8的最小rootfs，该脚本来源于IMG公司的CI20项目。
   因为所用硬件平台使用的调试串口为UART2，与CI20平台使用的调试串口不同，
   因此需要先进行相应修改，以下是脚本中的原始代码：

```
# Configure a serial console.
echo "T0:23:respawn:/sbin/getty -L ttyS0 115200 vt100" >> \
    "${target_dir}/etc/inittab"
```

复制代码

   修改后的代码如下：

```
# Configure a serial console.
echo "T0:23:respawn:/sbin/getty -L ttyS2 115200 vt100" >> \
    "${target_dir}/etc/inittab"
```

复制代码

   因为原脚本默认使用的是国外的源，为了避免因为网速问题导致打包时间过长，
   因此需要将其更改为国内的源，以下是脚本中的原始代码：

```
debian_packages="locales"
debian_mirror="http://ftp.uk.debian.org/debian/"
debian_path="/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin"
```

复制代码

   修改后的代码如下，可根据自己的实际情况选择其他的国内源：

```
debian_packages="locales"
debian_mirror="http://mirrors.ustc.edu.cn/debian/"
debian_path="/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin"
```

复制代码

2. 安装进行打包工作所必须的软件包。
   因mkdebianrfs.sh调用的是debootstrap，因此需要安装debootstrap。
   同时由于X1000E是MIPS指令集的处理器，为了在X86平台上进行打包，还需安装qemu模拟器。
   依次执行以下命令，完成对所需软件包的安装：

```
sudo apt-get update
sudo apt-get install binfmt-support qemu qemu-user-static debootstrap bzip2 gedit
```

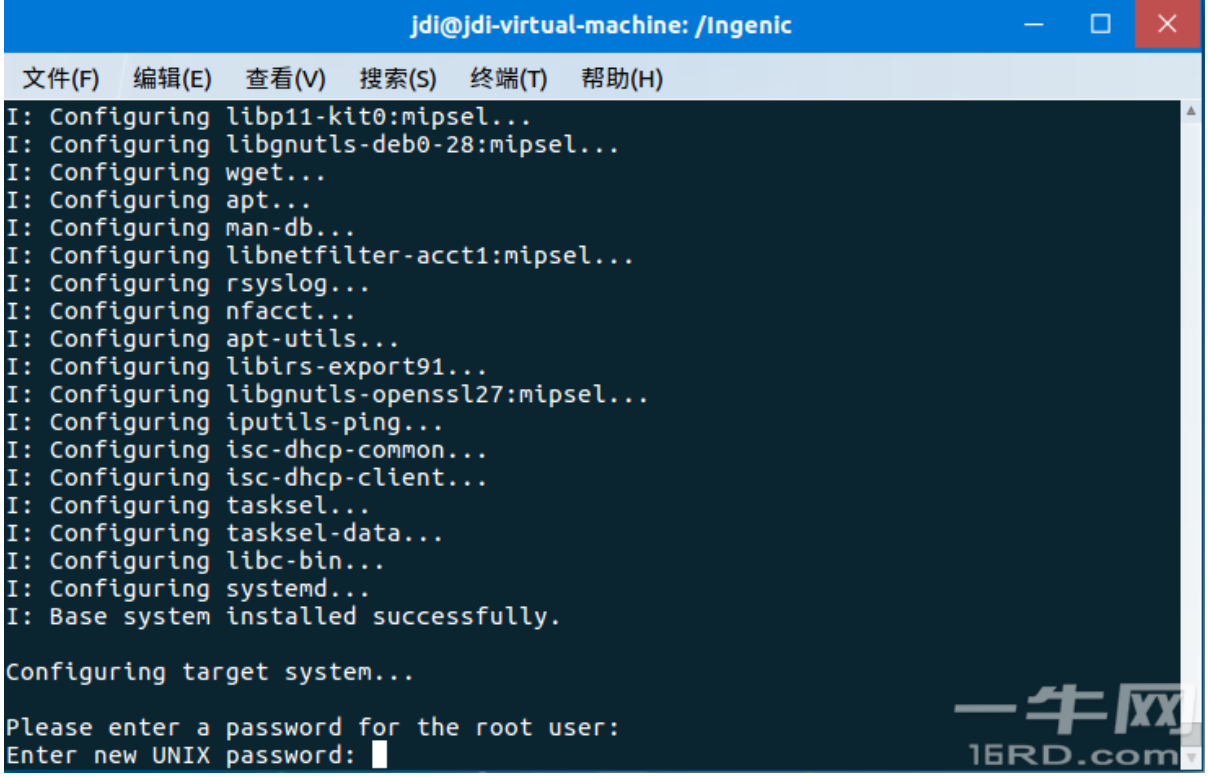
复制代码

3. 建立所需目录，并完成Debian8最小rootfs的打包及相关设置。
   依次执行以下命令创建所需目录并进行打包操作：

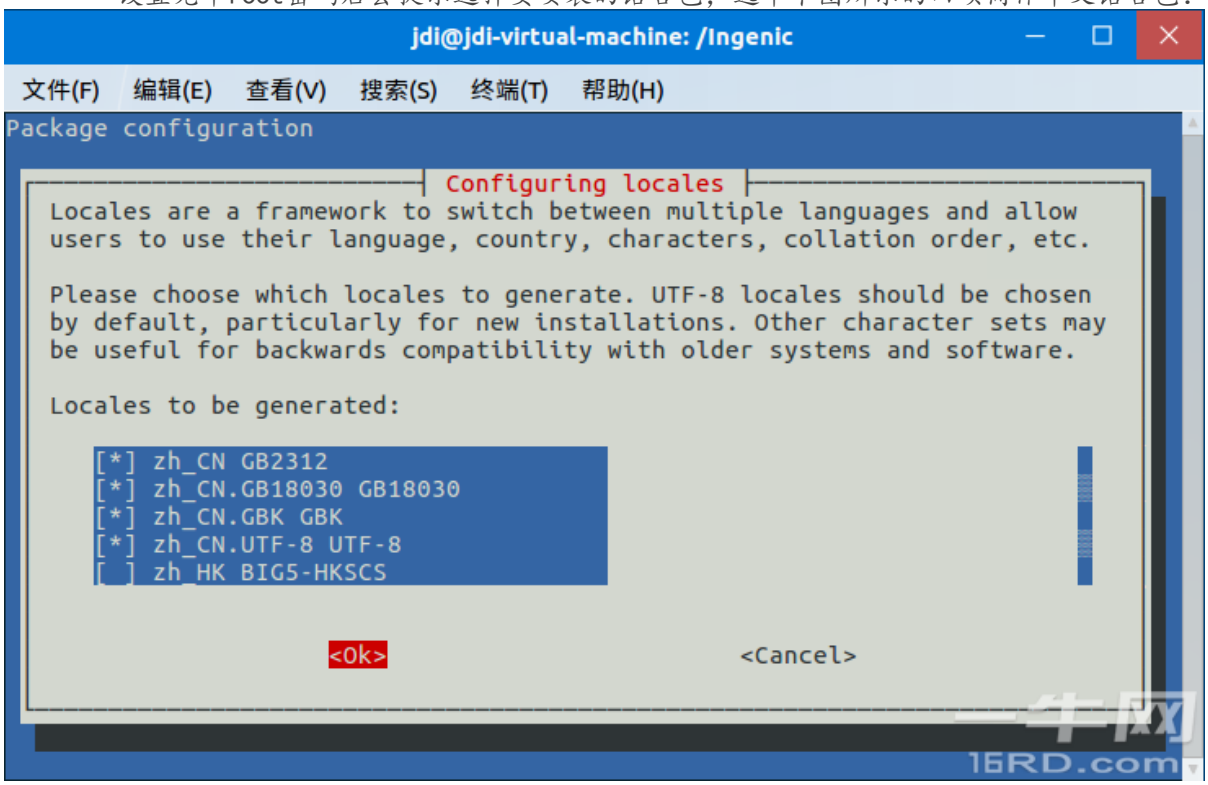
```
mkdir debian8-rootfs
sudo ./mkdebianrfs.sh mipsel jessie debian8-rootfs
```

复制代码

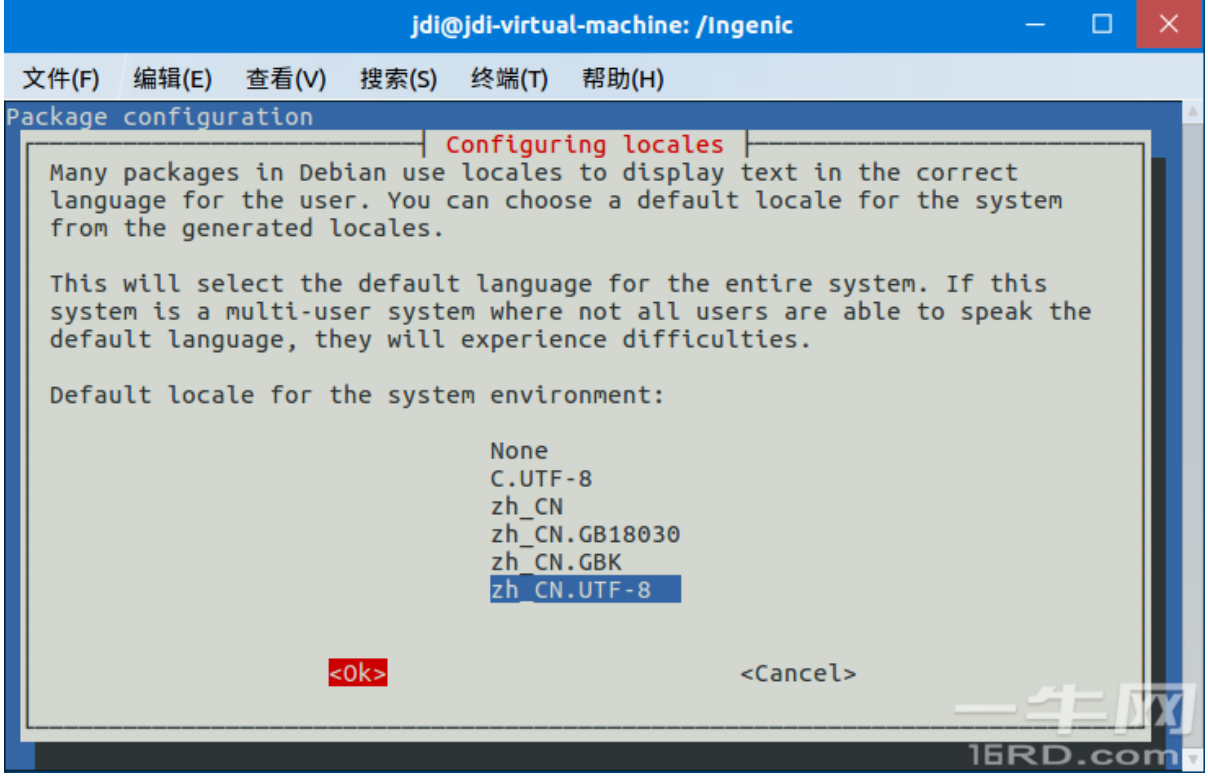
   此时将会开始进行Debian8最小rootfs的打包，根据网络情况耗时约几分钟至十几分钟。
   待打包完毕后会提示为root账户设置密码，如下图所示，共需输入两次。



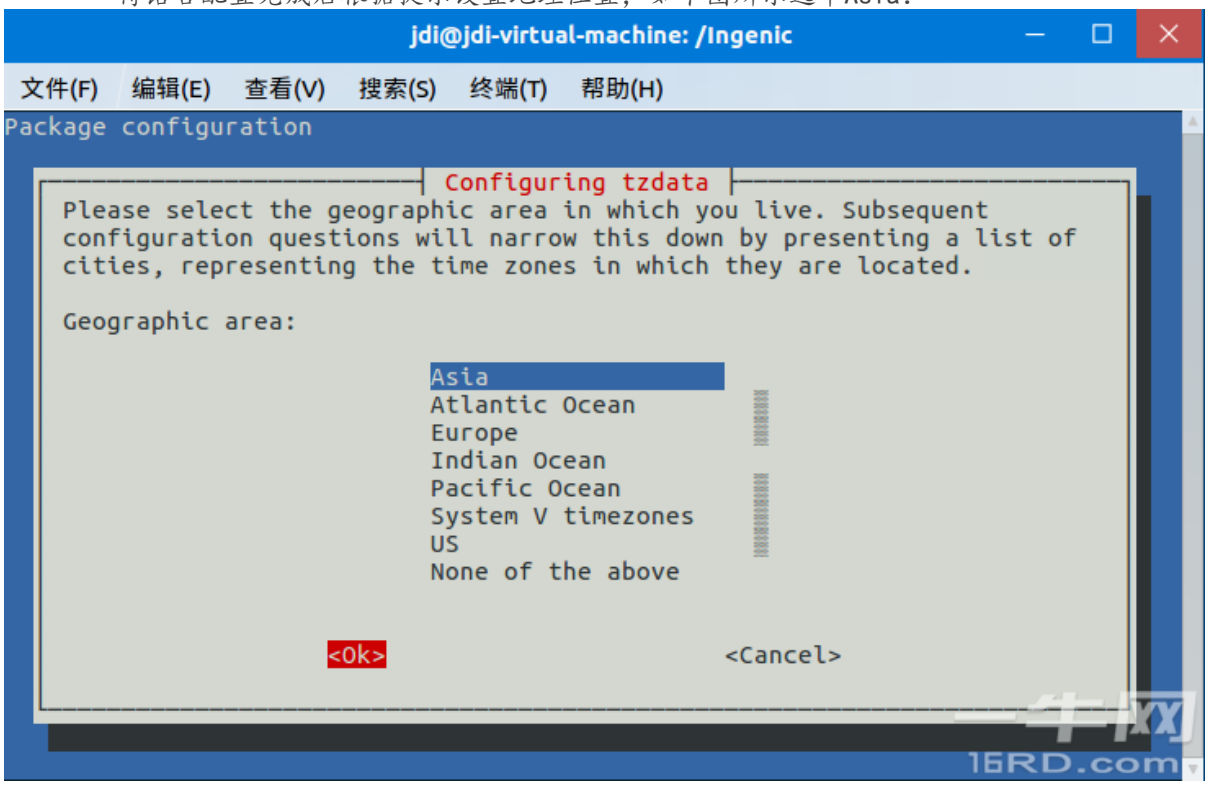

设置完毕root密码后会提示选择要安装的语言包，选中下图所示的四项简体中文语言包：




随后会提示选择系统默认使用的语言包，选中zh_CN.UTF-8，如下图所示。
此项步骤耗时较长，根据机器配置不同约需要3至5分钟。

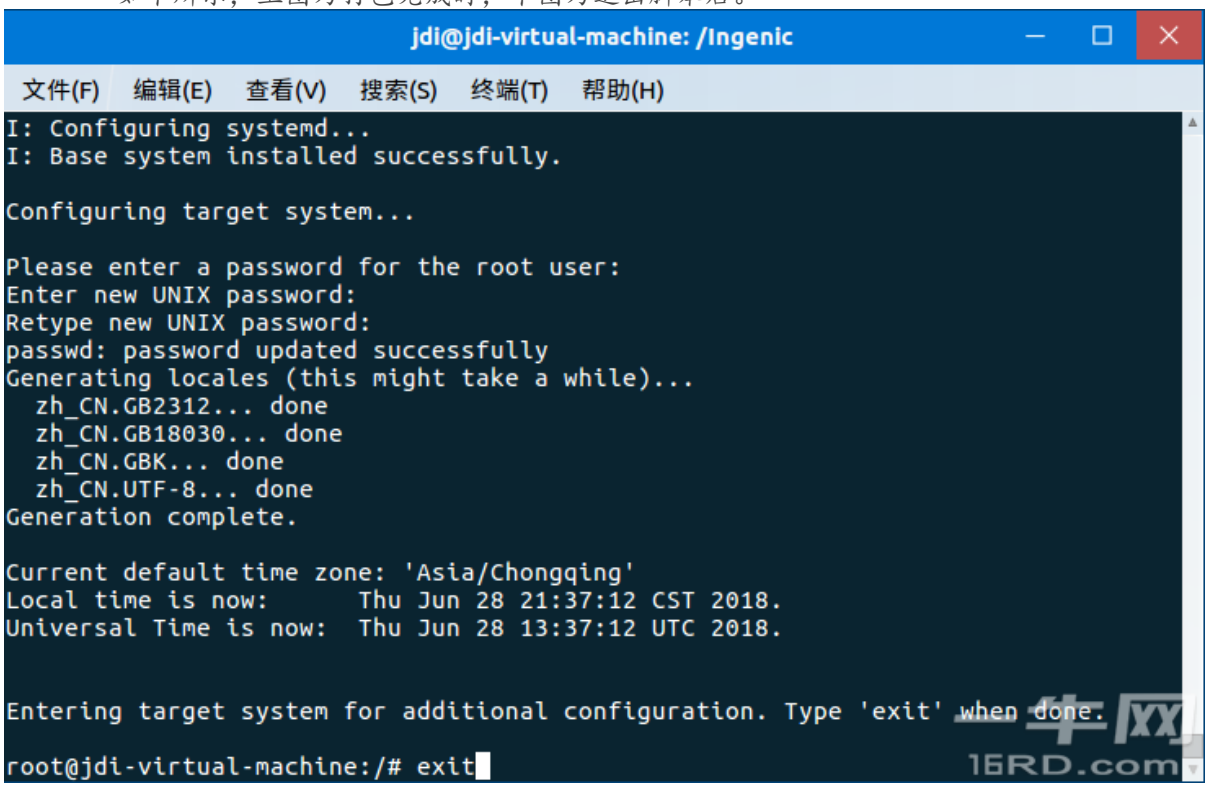


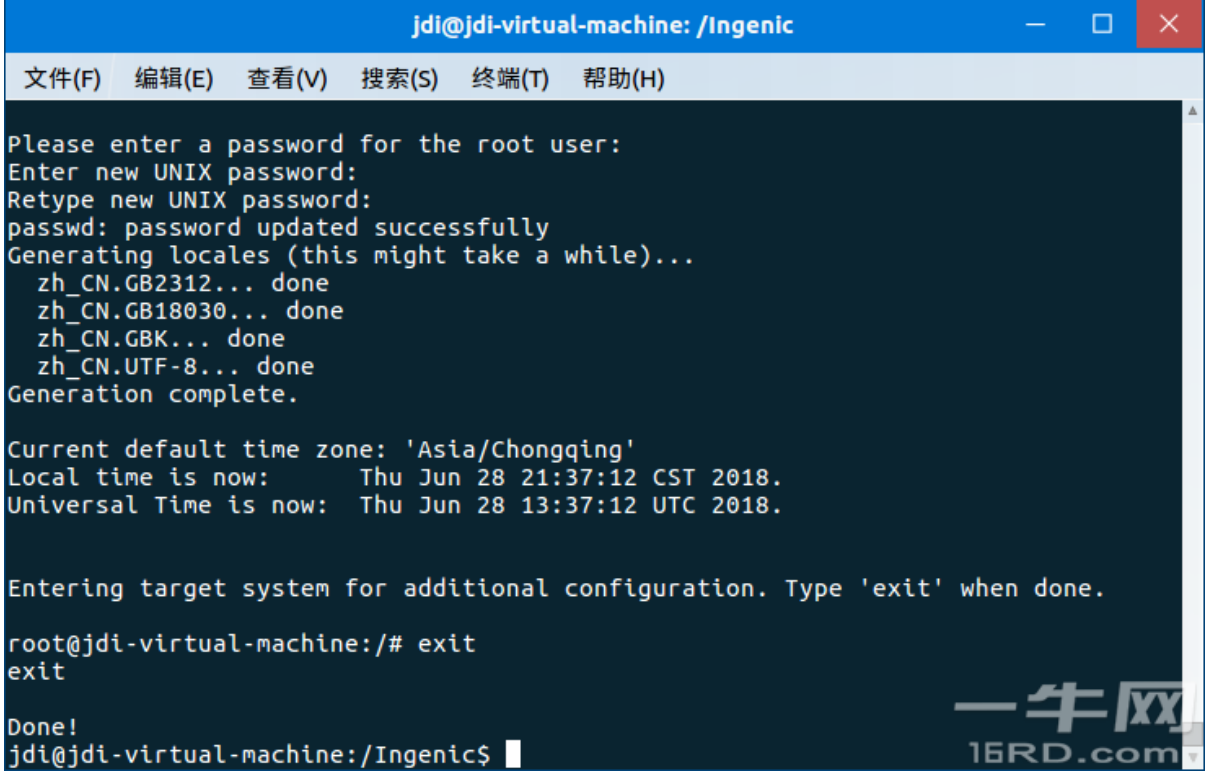

待语言配置完成后根据提示设置地理位置，如下图所示选中Asia:




根据提示设置时区，国内可选择上海或重庆，因本人在成都，所以就近选择重庆。


完成所有操作后，根据提示输入exit退出打包脚本，
如下所示，上图为打包完成时，下图为退出脚本后。






接下来就需要将打包完毕后的Debian8 Rootfs制作成ext4格式的镜像。
和制作Buildroot的ext4镜像类似，依次执行以下命令：

```
dd if=/dev/zero of=debian8-rootfs.ext4 bs=1M count=512
mkfs.ext4 debian8-rootfs.ext4
sudo mount -o loop debian8-rootfs.ext4 /mnt
sudo cp -a -fr debian8-rootfs/* /mnt
sync
sudo umount /mnt
sudo chmod 777 debian8-rootfs.ext4
```

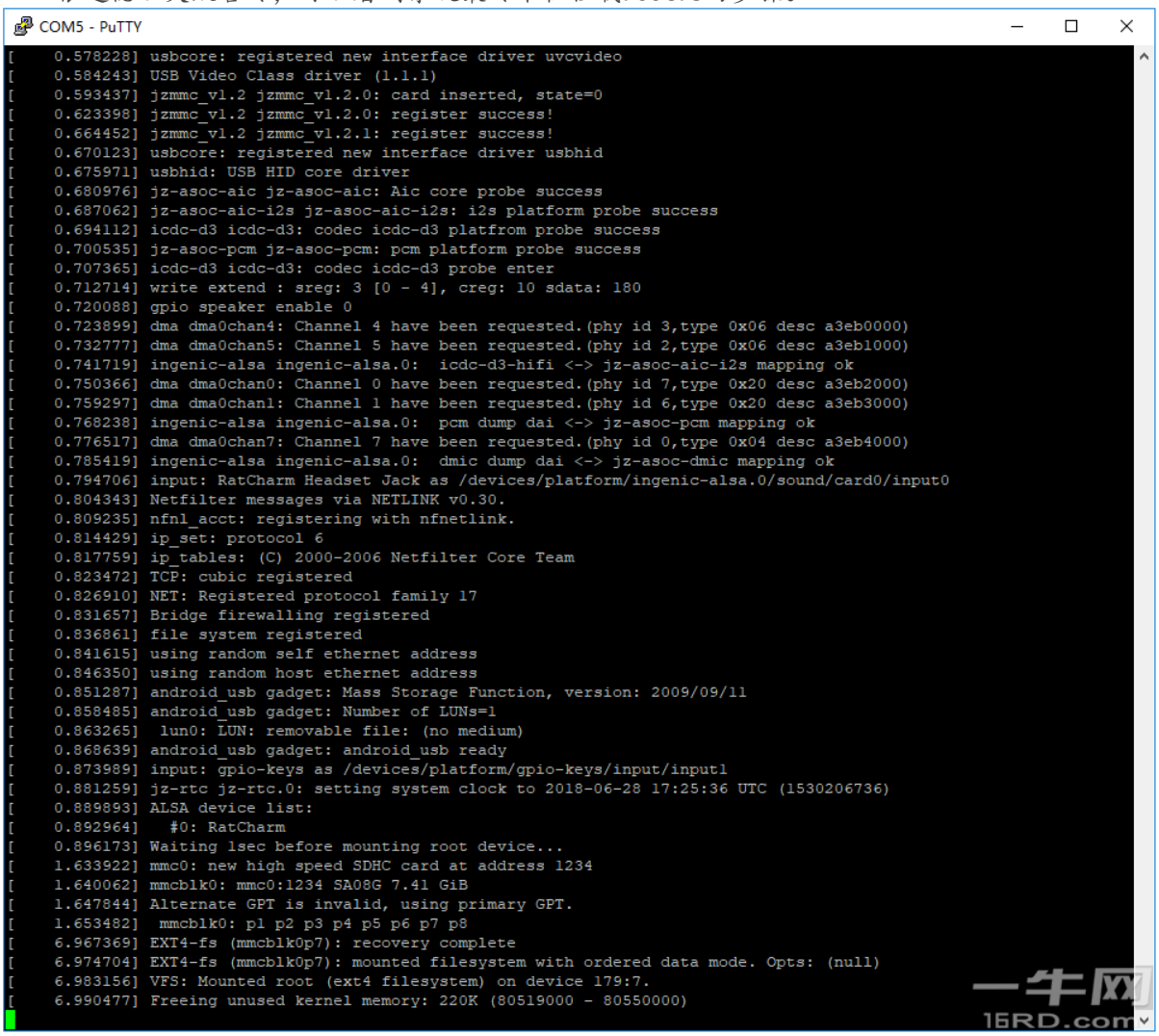
复制代码

若顺利完成则可得到名为debian8-rootfs.ext4的目标文件，
用该文件替换掉此前的system.ext4，使用cloner烧录到目标板。

## 接下来将尝试引导Debian8并解决相关问题

- 第一次尝试：

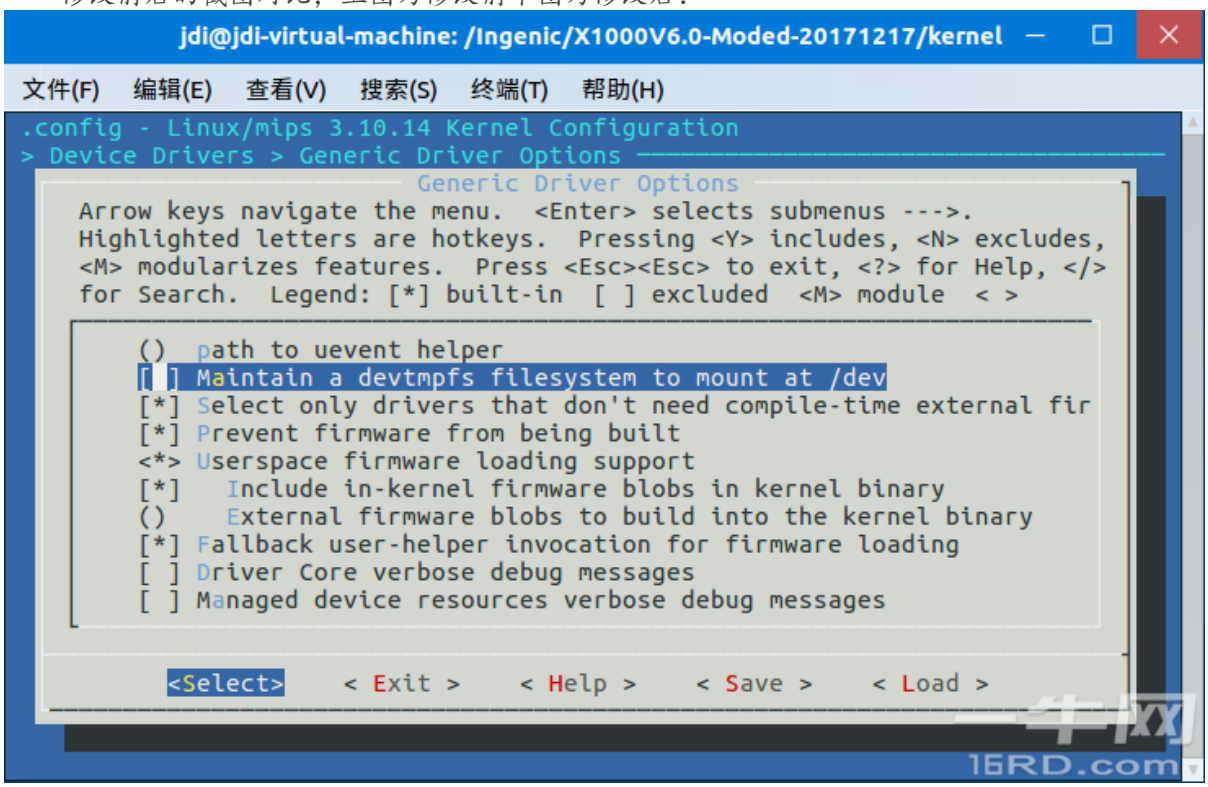
很遗憾以失败告终，可以看到系统最终卡在挂载rootfs的步骤。

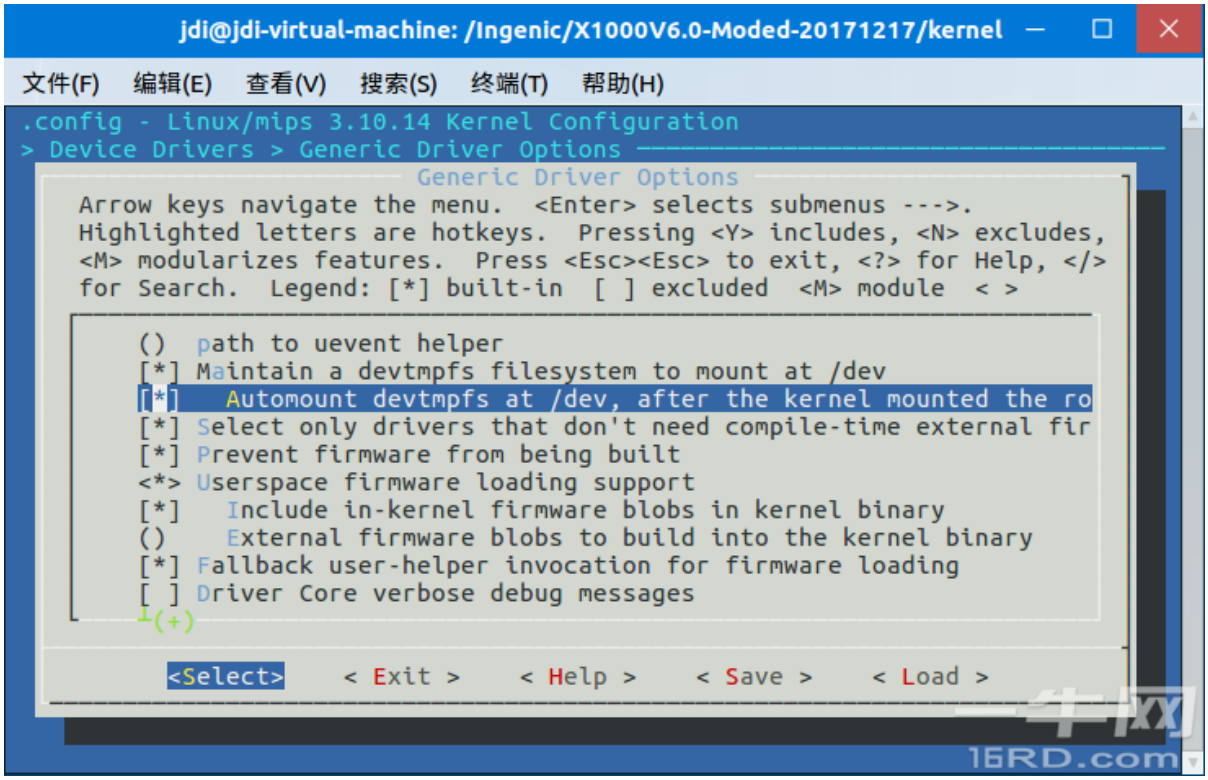



经分析，判断问题是因为内核中没有开启devtmpfs造成的，因此需要对kernel进行如下配置：

```
| Symbol: DEVTMPFS_MOUNT [=y]                                                 |
| Type  : boolean                                                             |
| Prompt: Automount devtmpfs at /dev, after the kernel mounted the rootfs     |
|   Location:                                                                 |
|     -> Device Drivers                                                       |
|       -> Generic Driver Options                                             |
|         -> Maintain a devtmpfs filesystem to mount at /dev (DEVTMPFS [=y]) |
|   Defined at drivers/base/Kconfig:44                                        |
|   Depends on: DEVTMPFS [=y]                                                 |
```

修改前后的截图对比，上图为修改前下图为修改后：

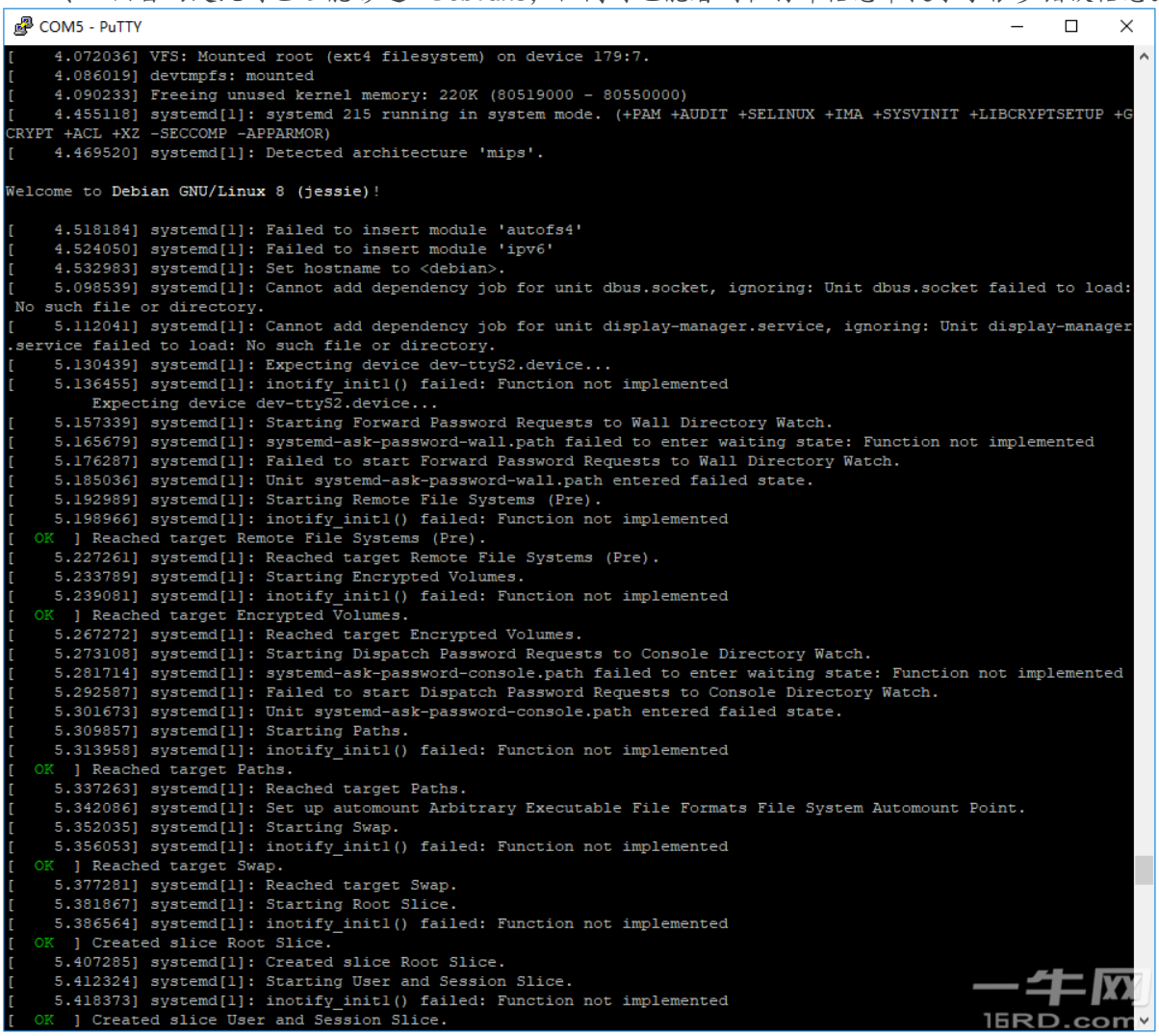



完成修改后重新编译kernel并烧写进目标板，准备进行第二次尝试。

- 第二次尝试：

令人兴奋的是此时已经能够进入Debian8，但同时也能看到在打印信息中提示了很多错误信息。




根据打印记录，梳理一下存在的问题：

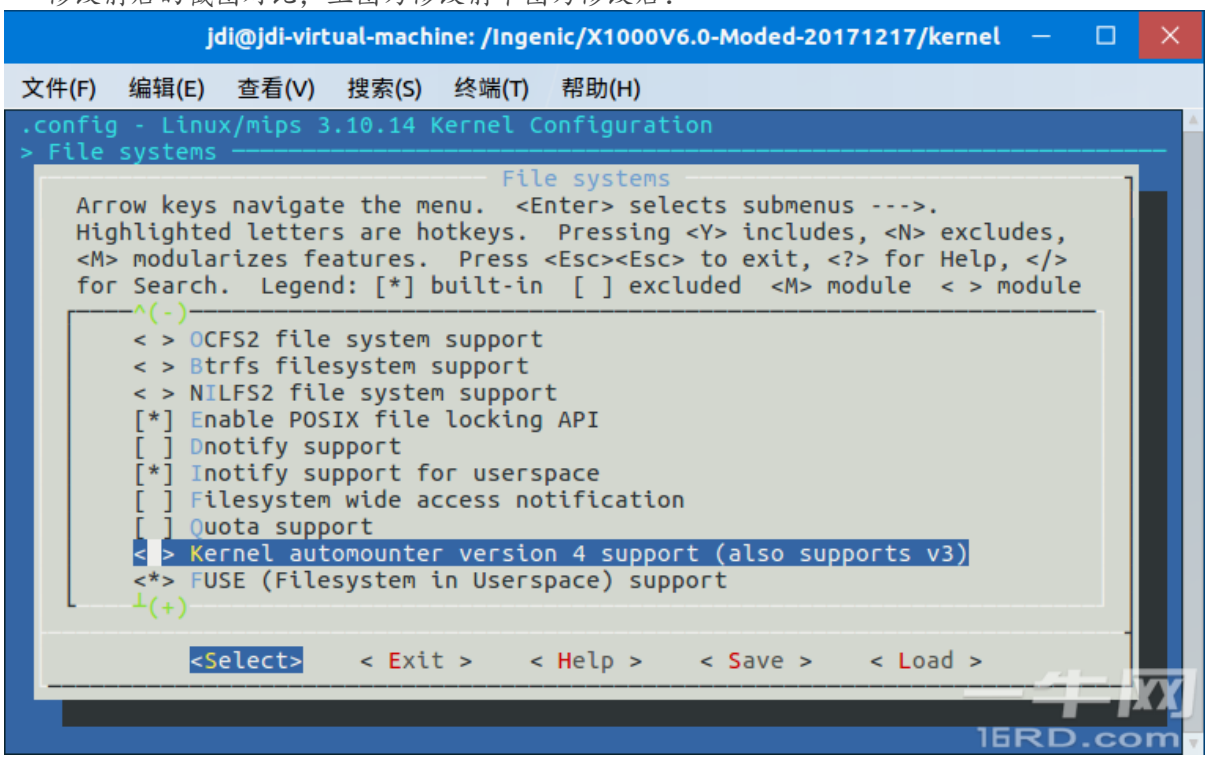
1. 与autofs4相关的问题；
2. 与ipv6相关的问题；
3. 与dbus相关的问题；
4. 与inotify相关的问题；
5. 与getty相关的问题(包括90秒超时和无法登陆两个问题)。

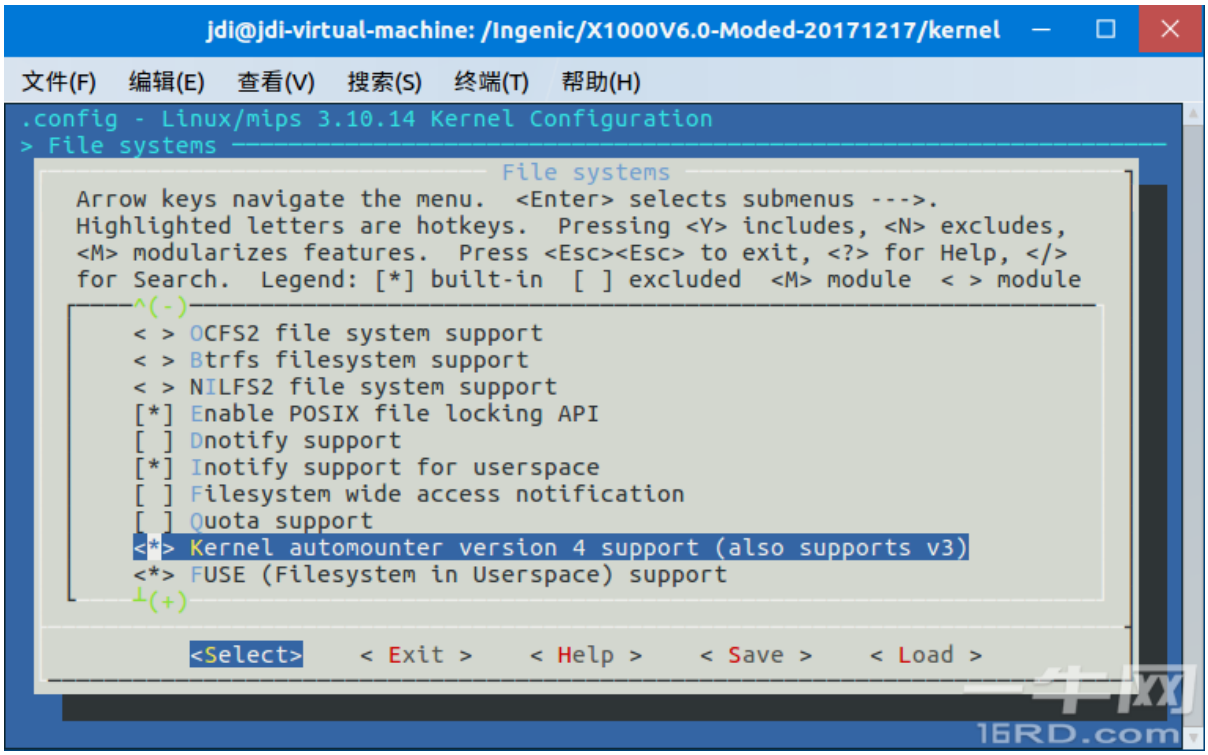
经过分析判定，与autofs4、ipv6、inotify相关的问题是因为kernel里没有开启相应的选项，需要对kernel进行对应配置。与dbus及getty相关的问题与rootfs有关，需要通过重新打包rootfs才能修复。本着先易后难的原则，先对kernel进行如下配置以便使能autofs4：

```
01.     | Symbol: AUTOFS4_FS [=y]                                              |
02.     | Type  : tristate                                                     |
03.     | Prompt: Kernel automounter version 4 support (also supports V3)      |
04.     |   Location:                                                          |
05.     |     -> File systems                                                  |
06.     |   Defined at fs/autofs4/Kconfig:1                                    |
```

复制代码

修改前后的截图对比，上图为修改前下图为修改后：

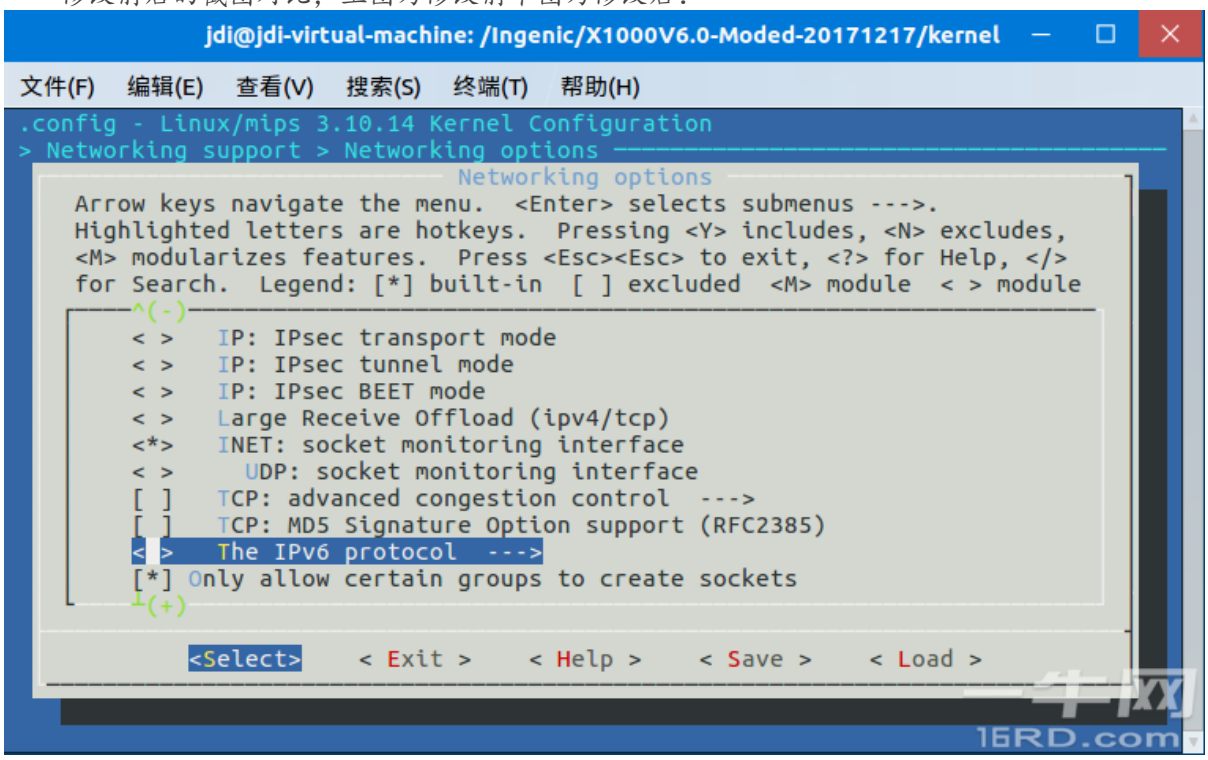


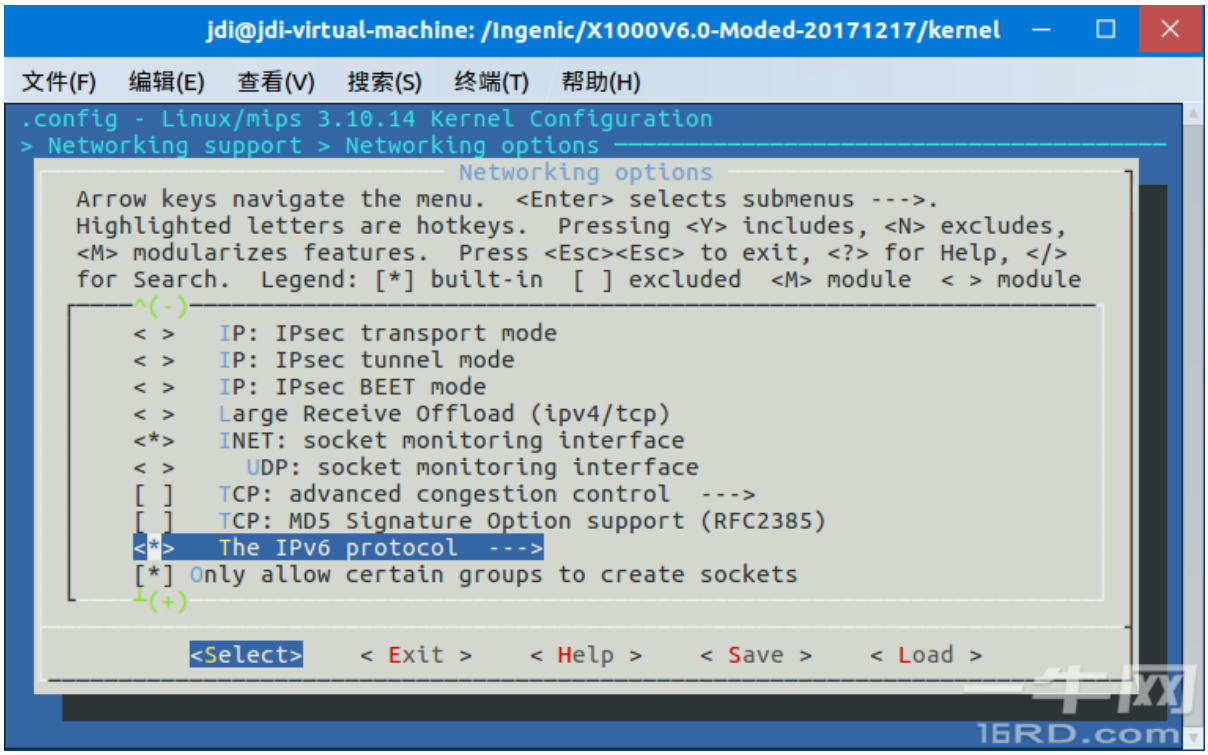




然后使能ipv6，需如下配置kernel：

```
01.     | Symbol: IPV6 [=y]                                                  |
02.     | Type  : tristate                                                   |
03.     | Prompt: The IPv6 protocol                                          |
04.     |   Location:                                                        |
05.     |     -> Networking support (NET [=y])                               |
06.     |       -> Networking options                                        |
07.     |         -> TCP/IP networking (INET [=y])                           |
08.     |   Defined at net/ipv6/Kconfig:6                                    |
09.     |   Depends on: NET [=y] && INET [=y]                                |
```

修改前后的截图对比，上图为修改前下图为修改后：

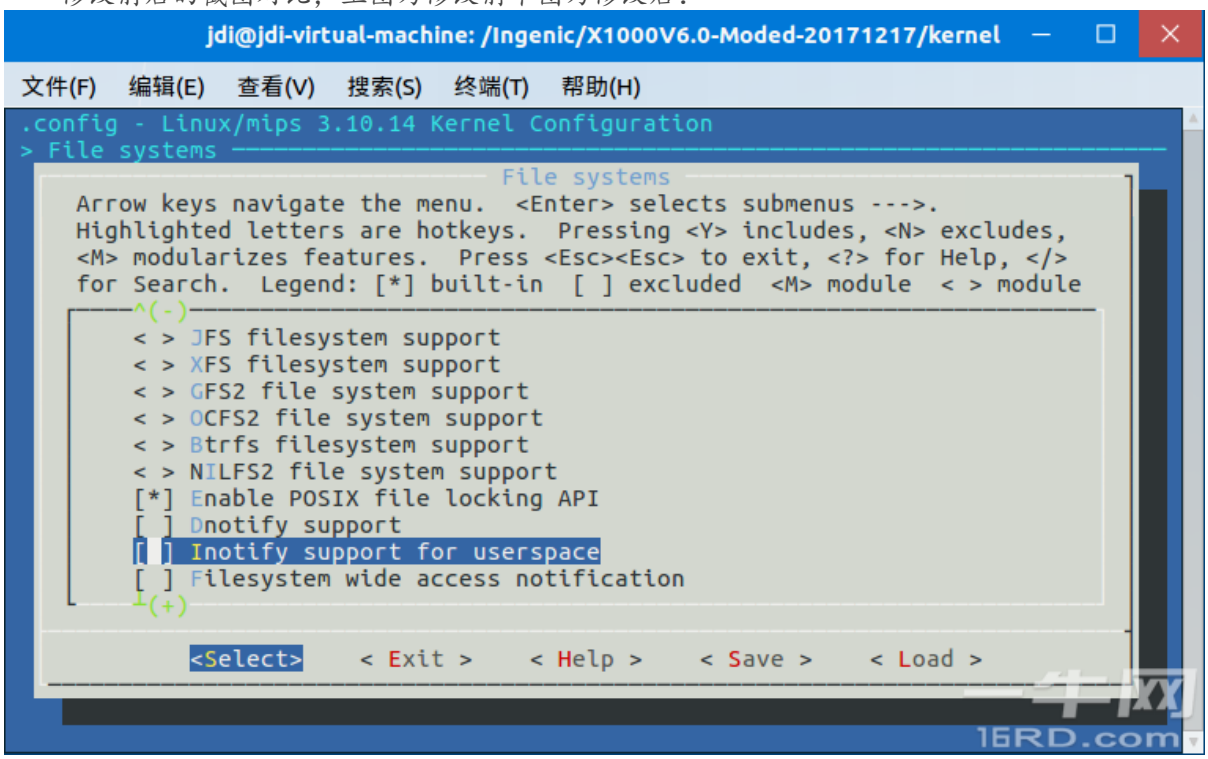


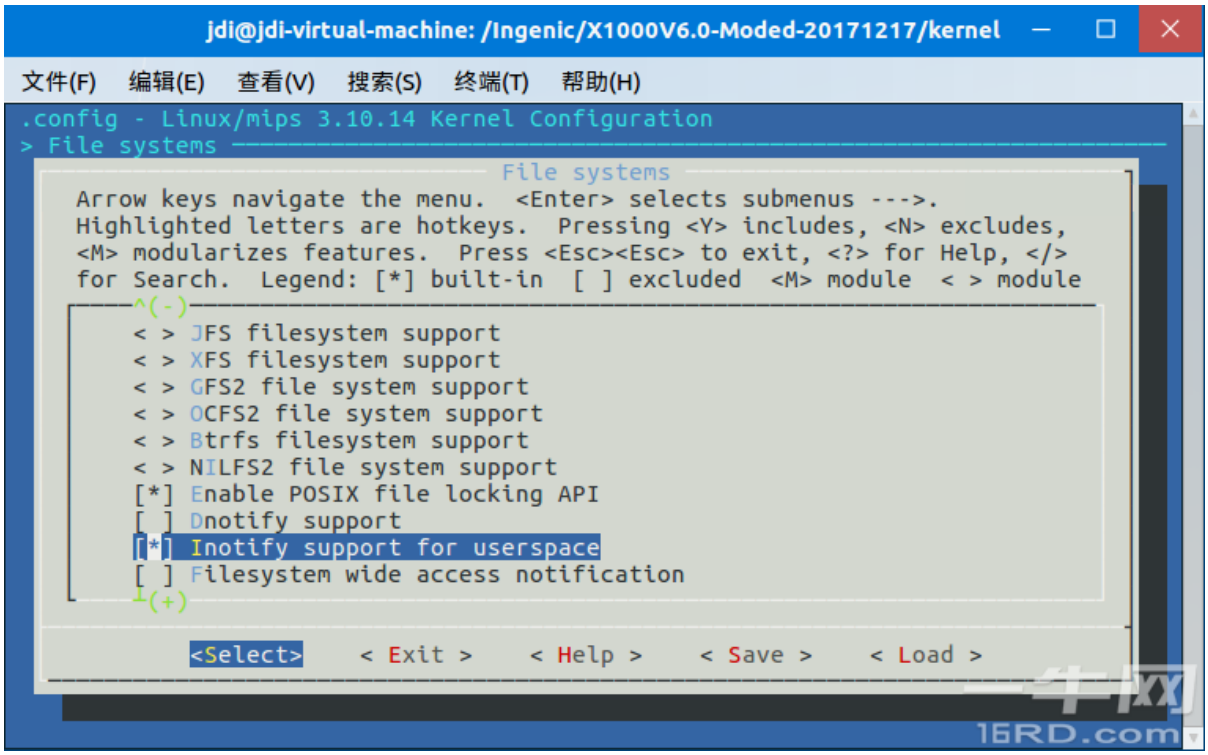




最后如下配置，以便解决inotify相关问题：

```
01.     | Symbol: INOTIFY_USER [=y]                                          |
02.     | Type  : boolean                                                    |
03.     | Prompt: Inotify support for userspace                              |
04.     |   Location:                                                        |
05.     |     -> File systems                                                |
06.     |   Defined at fs/notify/inotify/Kconfig:1                           |
07.     |   Selects: ANON_INODES [=y] && FSNOTIFY [=y]                       |
```

修改前后的截图对比，上图为修改前下图为修改后：

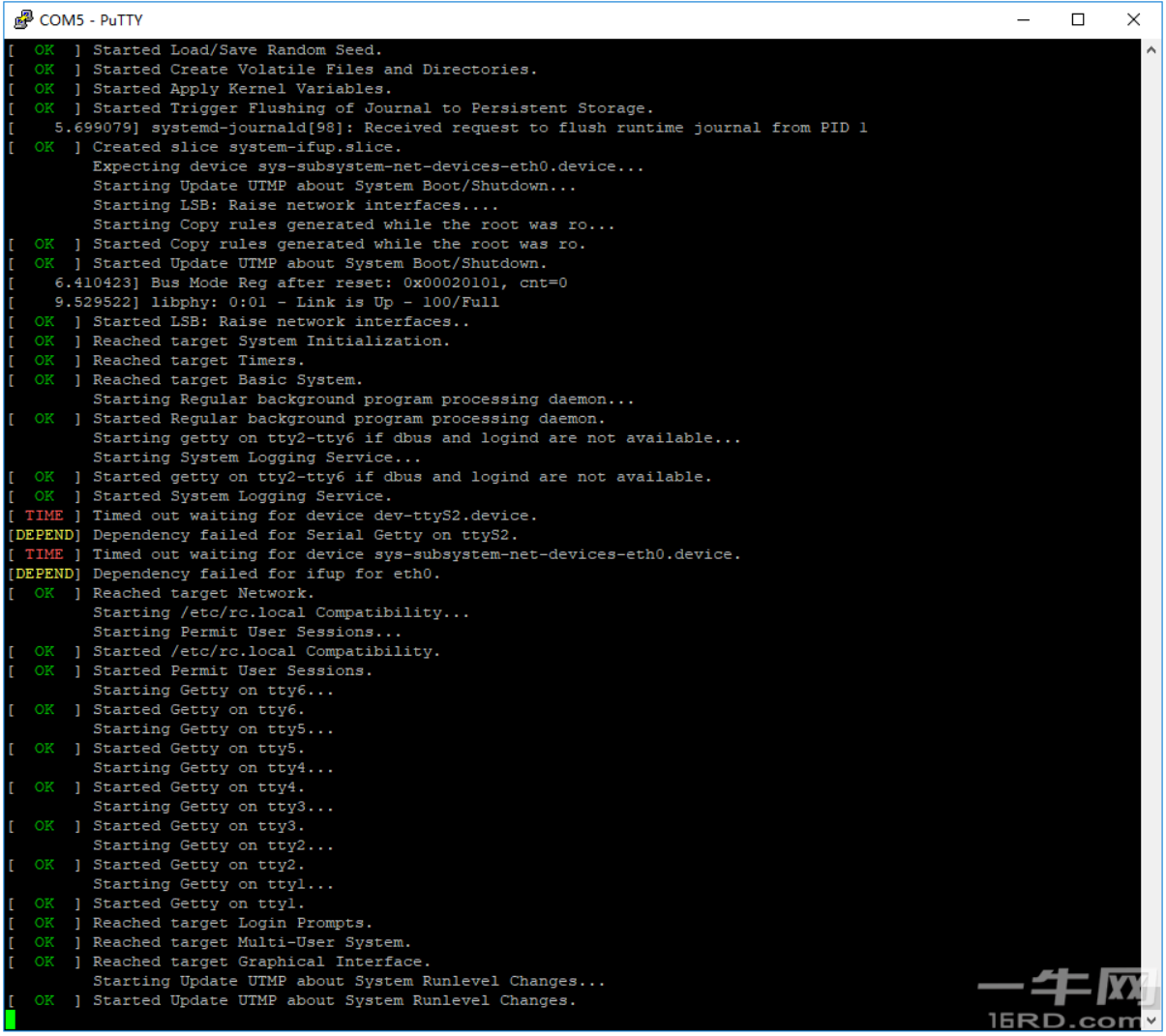



重新编译kernel并将其烧写到目标板，然后准备进行第三次引导尝试。

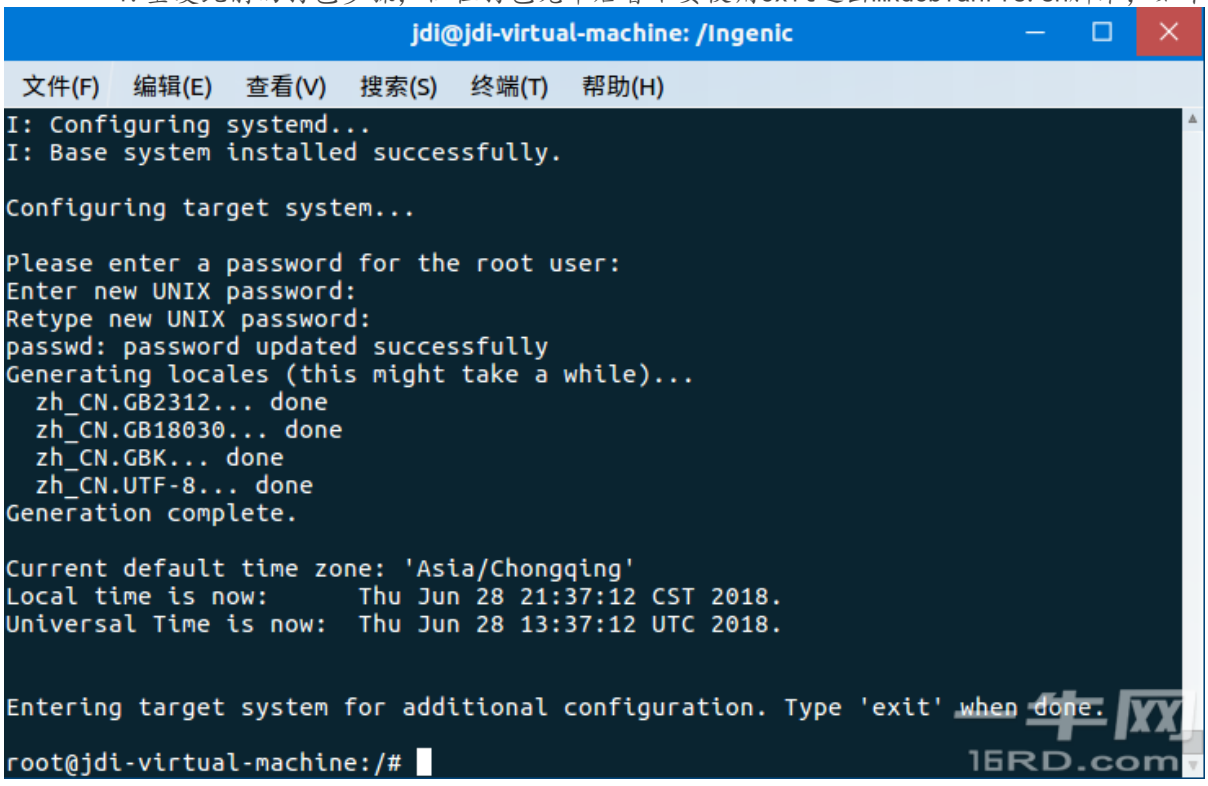
- 第三次尝试：

从打印log可以看到已经没有报与autofs4、ipv6及inotify相关的错误了，但又出现了一个新问题：
除了此前的getty存在90秒超时的问题，现在有线网络ifup也出现了90秒的超时。




新出现的ifup超时问题同此前的getty超时非常相似，分析后判断问题也出来rootfs中。
接下来将重新对Debian8的rootfs进行打包以便修复相关问题。操作步骤如下：

1. 重复此前的打包步骤，但在打包完毕后暂不要使用exit退出mkdebianrfs.sh脚本，如下图所示：




2. 新开启一个终端(以下称终端2)并进入到mkdebianrfs.sh脚本所在目录，执行以下命令挂载相应目录。

```
01.  sudo mount -t proc proc debian8-rootfs/proc
02.  sudo mount --bind /dev/pts/ debian8-rootfs/dev/pts/
```

复制代码

执行完以上两条命令后，若无错误提示则挂载成功，否则许根据提示排除问题后重新挂载。

3. 切换回之前打包的终端(以下称终端1)，依次输入以下命令安装所需软件包到打包好的rootfs中。

```
apt-get update
apt-get upgrade
apt-get install dbus vim
apt-get clean
```
复制代码

4. 创建getty所需的ttyS2配置文件解决getty超时和无法从串口登录的问题，在终端1中执行以下命令：

```
cp /lib/systemd/system/serial-getty\@.service /etc/systemd/system/getty.target.wants/serial-getty\@ttyS2.service
```
复制代码

执行完以上命令后，切换至终端2执行以下命令：

```
sudo gedit debian8-rootfs/etc/systemd/system/getty.target.wants/serial-getty@ttyS2.service
```
复制代码

在弹出的窗口中修改相关配置，以下是原始代码：

```
[Unit]
Description=Serial Getty on %I
Documentation=man:agetty(8) man:systemd-getty-generator(8)
Documentation=http://0pointer.de/blog/projects/serial-console.html
BindsTo=dev-%i.device
After=dev-%i.device systemd-user-sessions.service plymouth-quit-wait.service
After=rc-local.service
```
复制代码

修改后的代码如下：

```
[Unit]
Description=Serial Getty on %I
Documentation=man:agetty(8) man:systemd-getty-generator(8)
Documentation=http://0pointer.de/blog/projects/serial-console.html
After=systemd-user-sessions.service plymouth-quit-wait.service
After=rc-local.service
```
复制代码

完成修改并保存后，即可关闭窗口，至此即已完成对getty所需的ttyS2配置文件的创建和编辑。

5. 创建ifup所需的eth0配置文件解决ifup超时的问题，在终端1中依次执行以下命令：

```
mkdir /etc/systemd/system/network.target.wants
cp /lib/systemd/system/ifup\@.service /etc/systemd/system/network.target.wants/ifup\@eth0.service
```
复制代码

执行完以上命令后，切换至终端2执行以下命令：

```
sudo gedit debian8-rootfs/etc/systemd/system/network.target.wants/ifup@eth0.service
```
复制代码

在弹出的窗口中修改相关配置，以下是原始代码：

```
[Unit]
```

```
02.  Description=ifup for %I
03.  After=local-fs.target network-pre.target networking.service systemd-sysctl.service
04.  Before=network.target
05.  BindsTo=sys-subsystem-net-devices-%i.device
06.  After=sys-subsystem-net-devices-%i.device
07.  ConditionPathIsDirectory=/run/network
08.  DefaultDependencies=no
```
复制代码

修改后的代码如下：

```
01.  [Unit]
02.  Description=ifup for %I
03.  After=local-fs.target network-pre.target networking.service systemd-sysctl.service
04.  Before=network.target
05.  ConditionPathIsDirectory=/run/network
06.  DefaultDependencies=no
```
复制代码

完成修改并保存后，即可关闭窗口，至此即已完成对ifup所需的eth0配置文件的创建和编辑。

6. 在终端1中使用exit退出脚本然后切换至终端2，依次输入以下命令卸载掉之前挂载的目录。

```
01.  sudo umount debian8-rootfs/proc
02.  sudo umount debian8-rootfs/dev/pts/
```
复制代码

若在卸载debian8-rootfs/proc时出现类似以下的错误：

```
01.  umount: /Ingenic/debian8-rootfs/proc: target is busy
02.          (In some cases useful info about processes that
03.           use the device is found by lsof(8) or fuser(1).)
```
复制代码

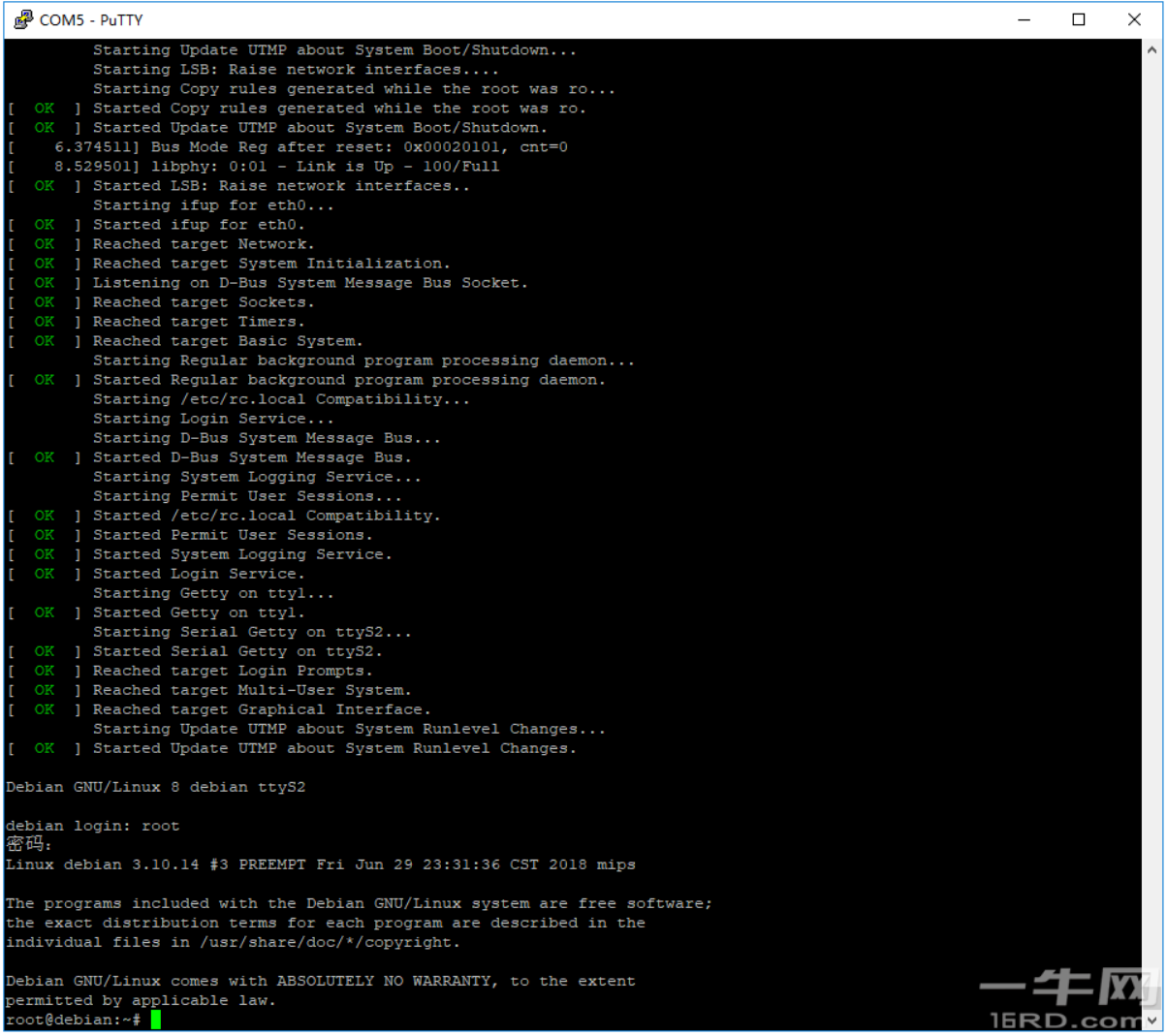
则首先确保没有打开与要卸载目录有关的文件夹或文件，然后使用以下命令强制卸载：

```
01.  sudo umount -lf debian8-rootfs/proc
```
复制代码

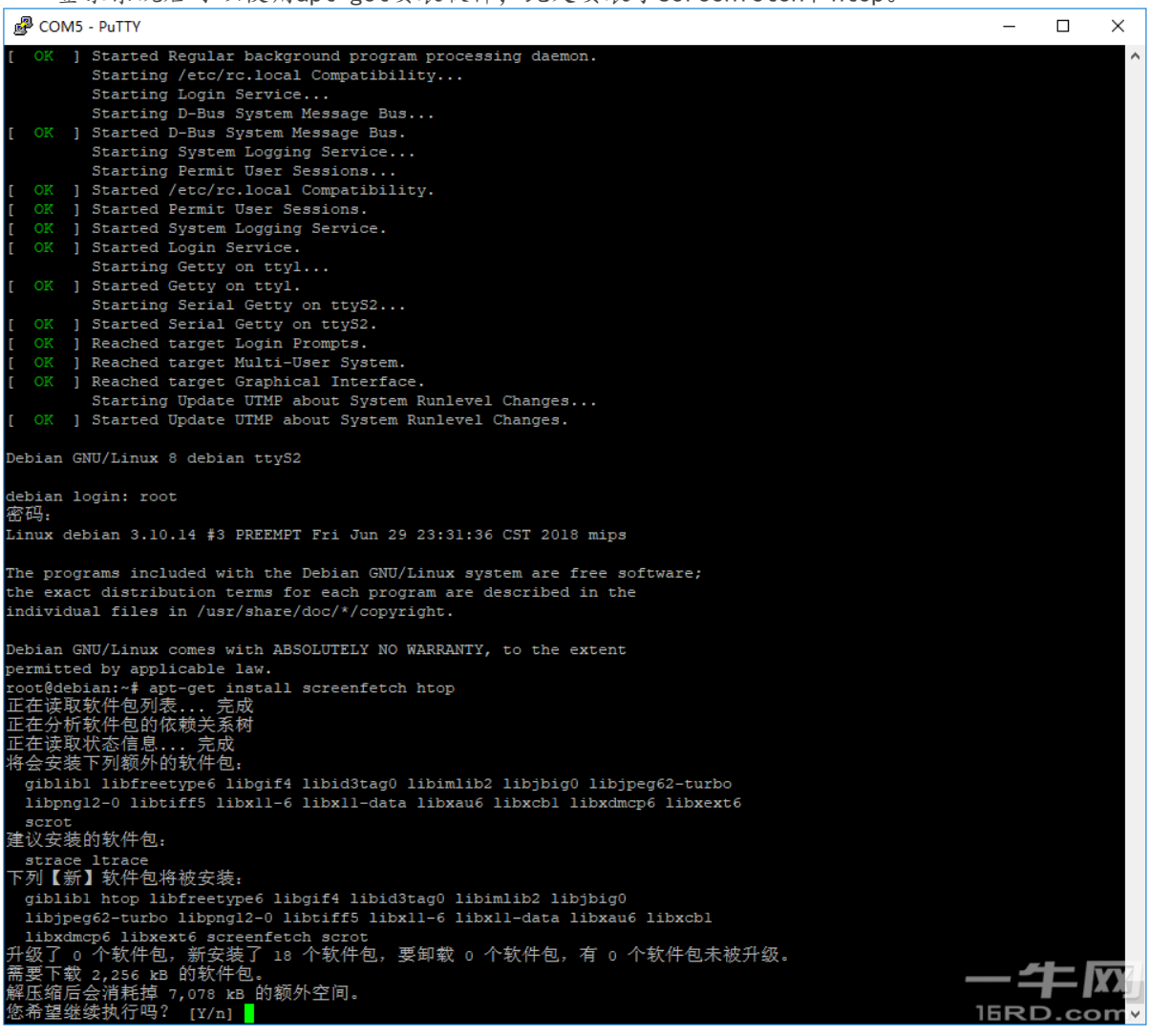
完成卸载后关掉终端2，按照此前制作ext4格式镜像的步骤重新制作镜像，
然后将新镜像烧录到目标板，进行新一轮的引导尝试。

- **第四次尝试：**

从打印log可以看到前面的问题都已经解决，并且已经出现了Debian8的登录提示，
并且使用root账户和此前打包时设置的密码，可以成功登录到系统。

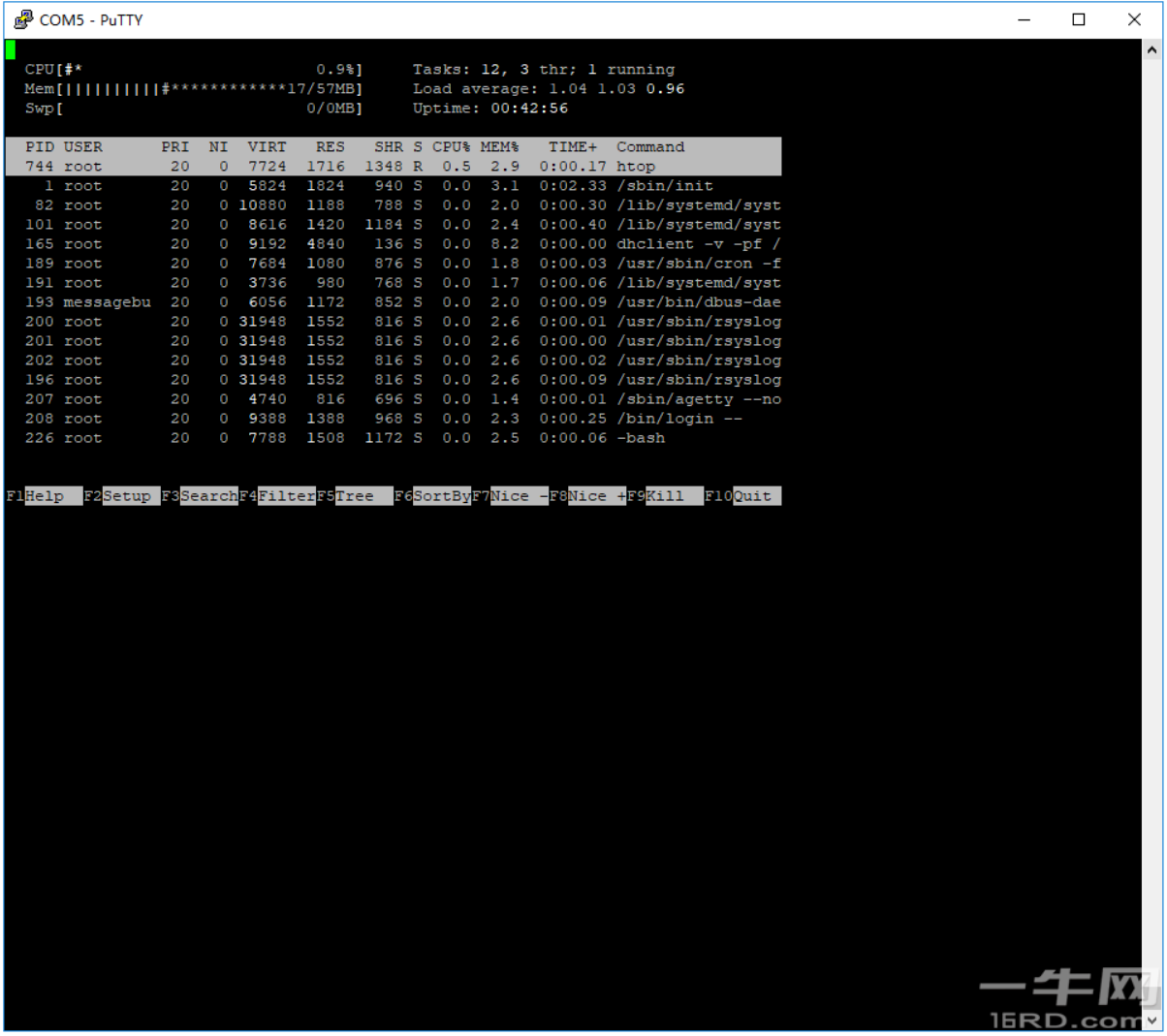



登录系统后可以使用apt-get安装软件，此处安装了screenfetch和htop。

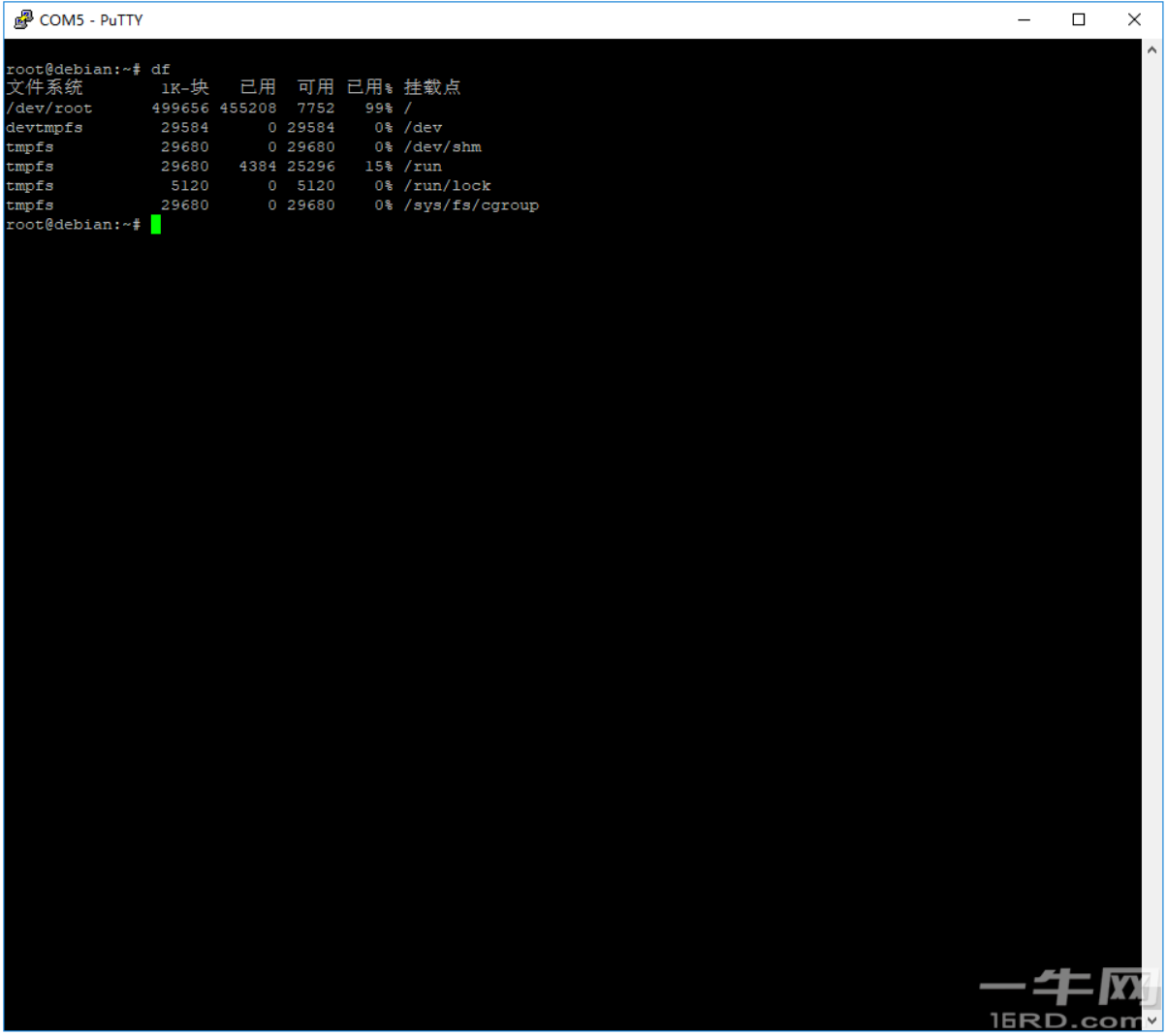


安装完毕后运行screenfetch，可以识别出各种系统信息。
比较有意思的是处理器被识别成了IBM的PowerPC。


运行htop，可以查看CPU占用率、内存占用率、SWAP交换文件占用率、当前运行的进程等信息。




由于X1000E仅有64M内存，设置一个较大的SWAP交换区就显得非常重要。
但打包好的rootfs默认是没有SWAP交换区的，这一点从htop中SWAP为0也可以证实。
因此接下来将尝试为系统添加SWAP分区，首先使用df命令查看当前可用的存储空间。

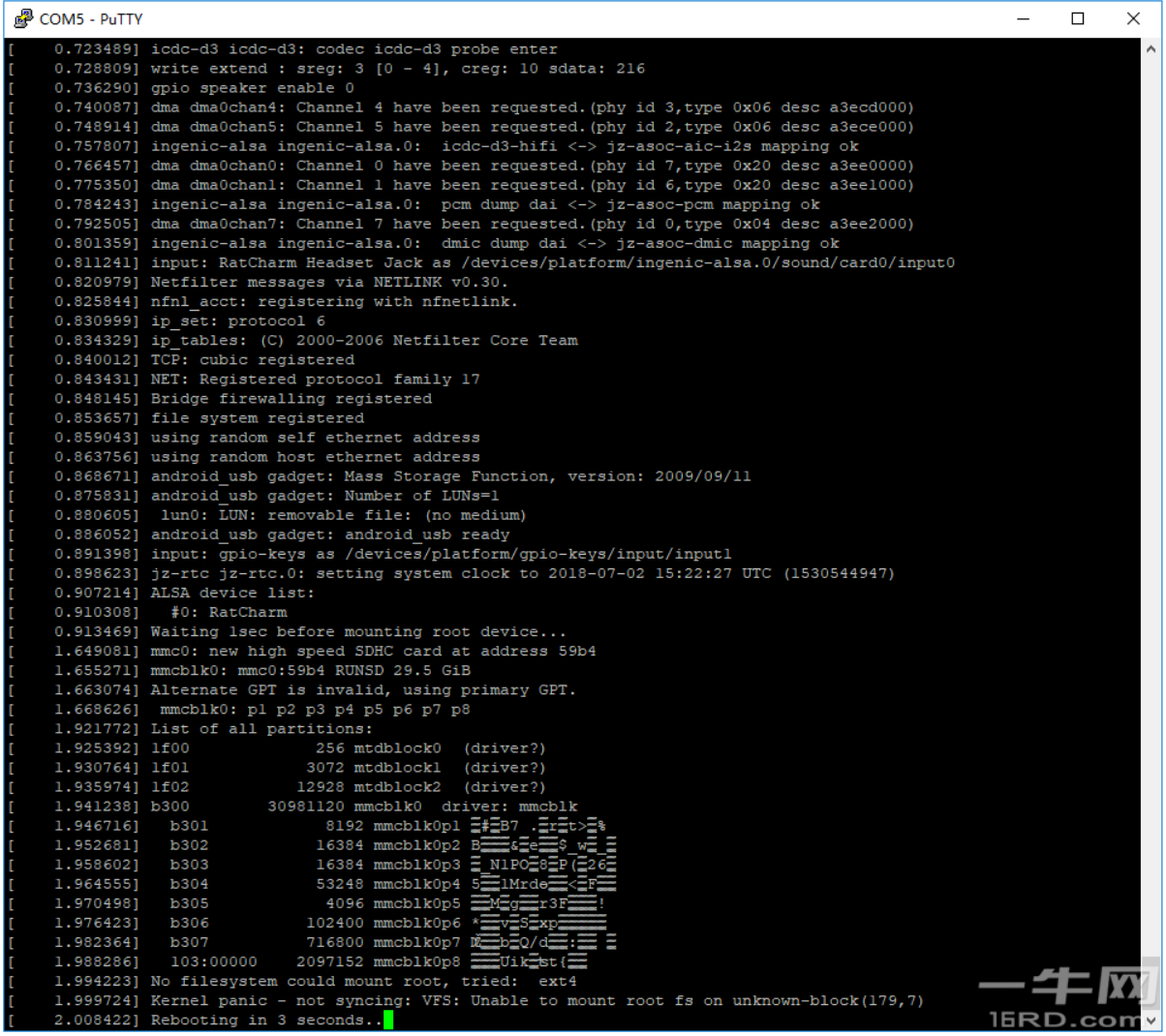



从返回的信息可以看见，rootfs空间占用已达到99%，仅剩约7MB的剩余空间。
所以需要打包一份更大的rootfs镜像，考虑到使用的TF卡容量足够大，
决定将镜像大小从512MB扩大到4GB，依次执行以下命令：

```
dd if=/dev/zero of=debian8-rootfs.ext4 bs=1M count=4096
mkfs.ext4 debian8-rootfs.ext4
sudo mount -o loop debian8-rootfs.ext4 /mnt
sudo cp -a -fr debian8-rootfs/* /mnt
sync
sudo umount /mnt
sudo chmod 777 debian8-rootfs.ext4
```

复制代码

将新制作好的ext4镜像烧录到目标板，然而很遗憾，出现了kernel panic的错误：




根据kernel panic的提示，经分析应该是4GB的rootfs大小超过了分区表允许的大小。
为了解决这个问题，需要对u-boot中涉及到分区配置的文件进行修改。
对u-boot目录中的board/ingenic/RatCharm/partitions.tab进行修改，以下是原始代码：

```
property:
    disk_size = 4096m
    gpt_header_lba = 512
    custom_signature = 0

partition:
        #name     =  start,   size, fstype
        xboot     =     0m,     3m,
        boot      =     3m,     8m, EMPTY
        recovery  =    12m,    16m, EMPTY
        pretest   =    28m,    16m, EMPTY
        reserved  =    44m,    52m, EMPTY
        misc      =    96m,     4m, EMPTY
        cache     =   100m,   100m, LINUX_FS
        system    =   200m,   700m, LINUX_FS
        data      =   900m,  2048m, LINUX_FS

#fstype could be: LINUX_FS, FAT_FS, EMPTY
```

复制代码

修改后的代码如下：

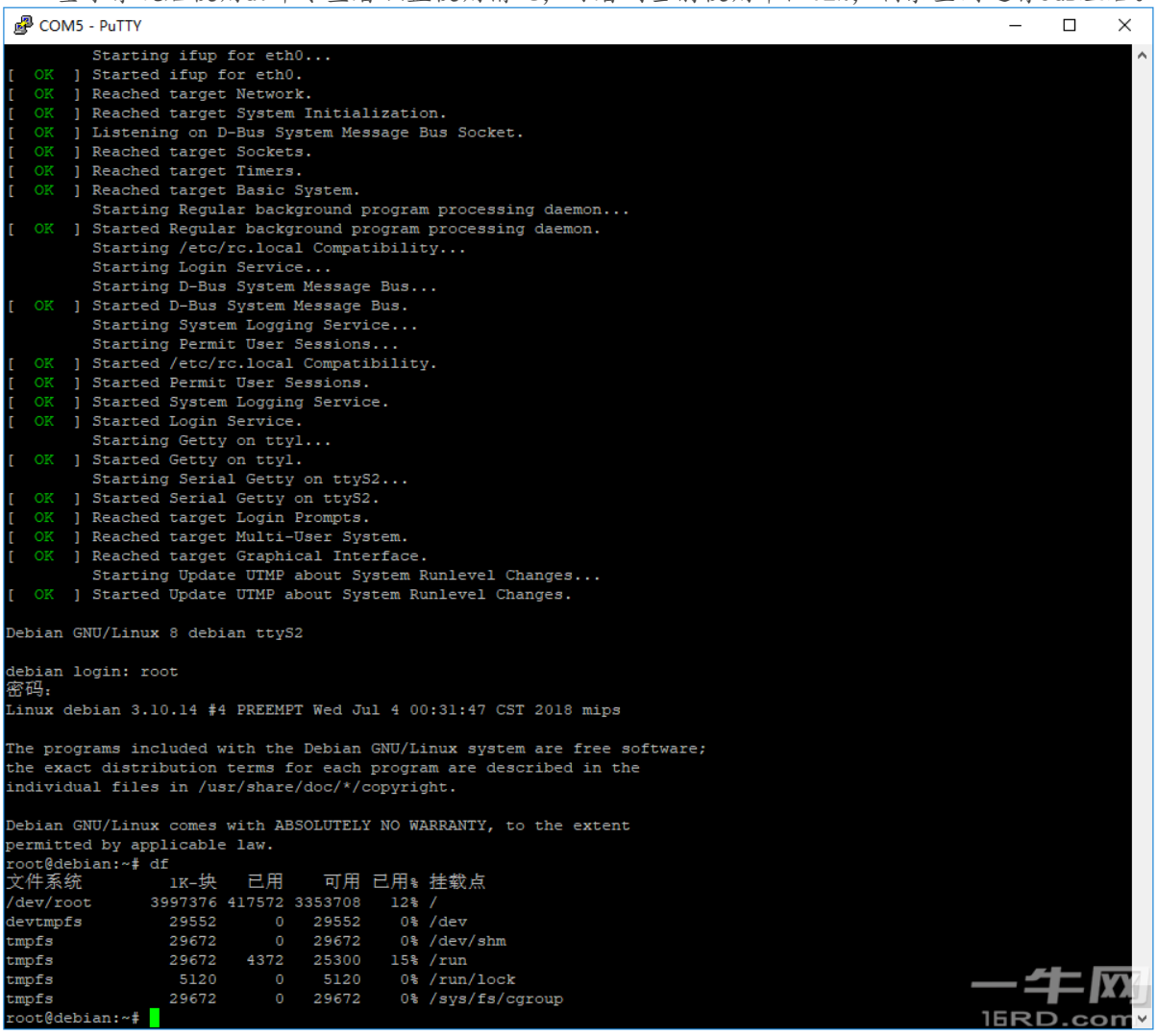
```
property:
    disk_size = 8192m
    gpt_header_lba = 512
```

```
    custom_signature = 0

partition:
        #name     =  start,   size, fstype
        xboot     =     0m,     3m,
        boot      =     3m,     8m, EMPTY
        recovery  =    12m,    16m, EMPTY
        pretest   =    28m,    16m, EMPTY
        reserved  =    44m,    52m, EMPTY
        misc      =    96m,     4m, EMPTY
        cache     =   100m,   100m, LINUX_FS
        system    =   200m,  5000m, LINUX_FS
        data      =  5200m,  2048m, LINUX_FS

#fstype could be: LINUX_FS, FAT_FS, EMPTY
```
复制代码

修改完毕后重新编译u-boot(编译前务必用make distclean清理)并下载至目标板，然后再次尝试引导系统。

- **第五次尝试：**

登录系统后使用df命令查看磁盘使用情况，可看到当前使用率仅12%，剩余空间还有3GB左右。




接下来准备设置SWAP交换区，依次输入以下命令：

```
mkdir /opt/swap
dd if=/dev/zero of=/opt/swap/swapfile bs=1M count=256
mkswap /opt/swap/swapfile
chmod 0600 /opt/swap/swapfile
```
复制代码

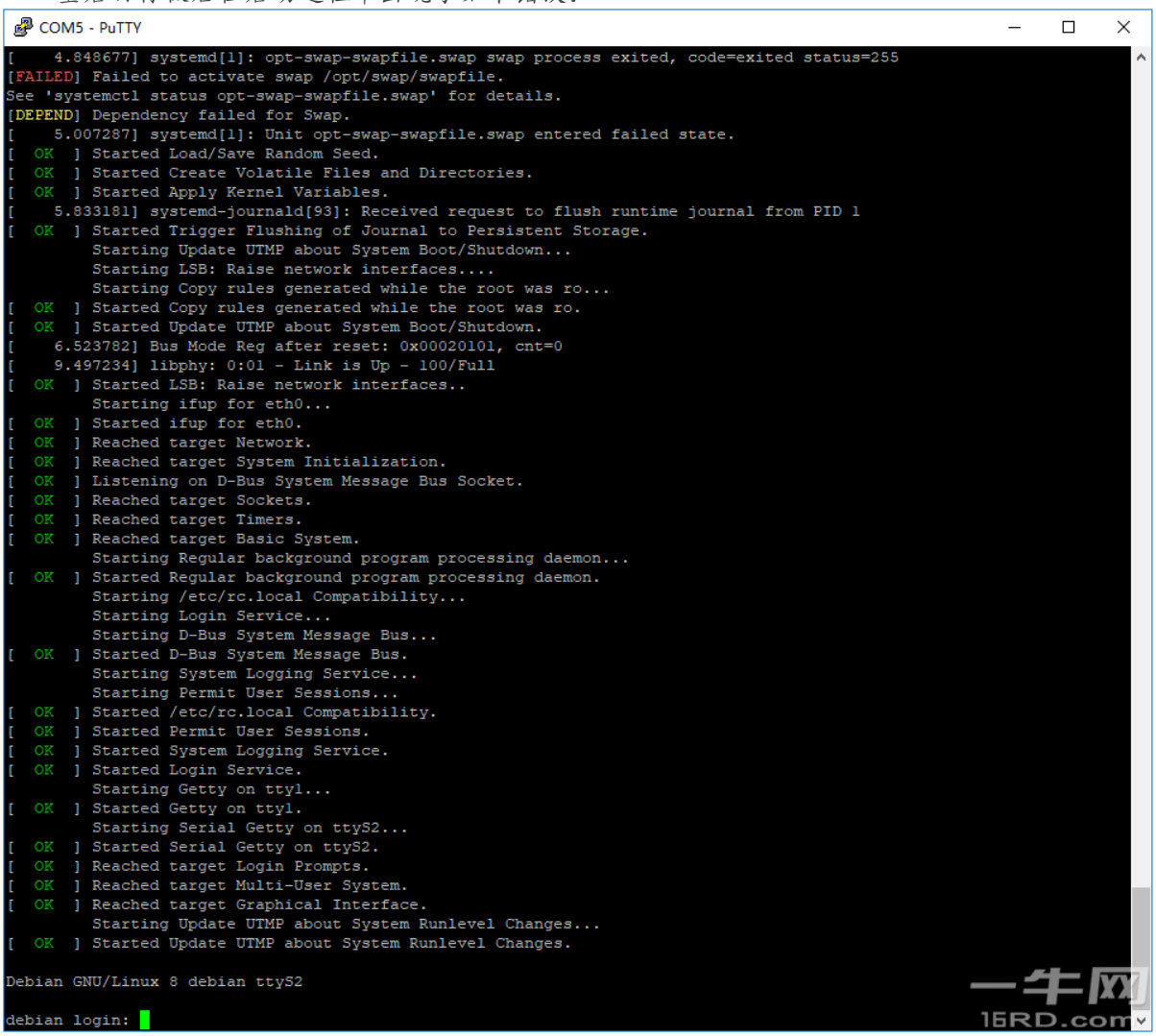
配置好swapfile文件后，我们还需要在/etc/fstab中添加相应命令，输入以下命令：

```
01.  vim /etc/fstab
```
复制代码

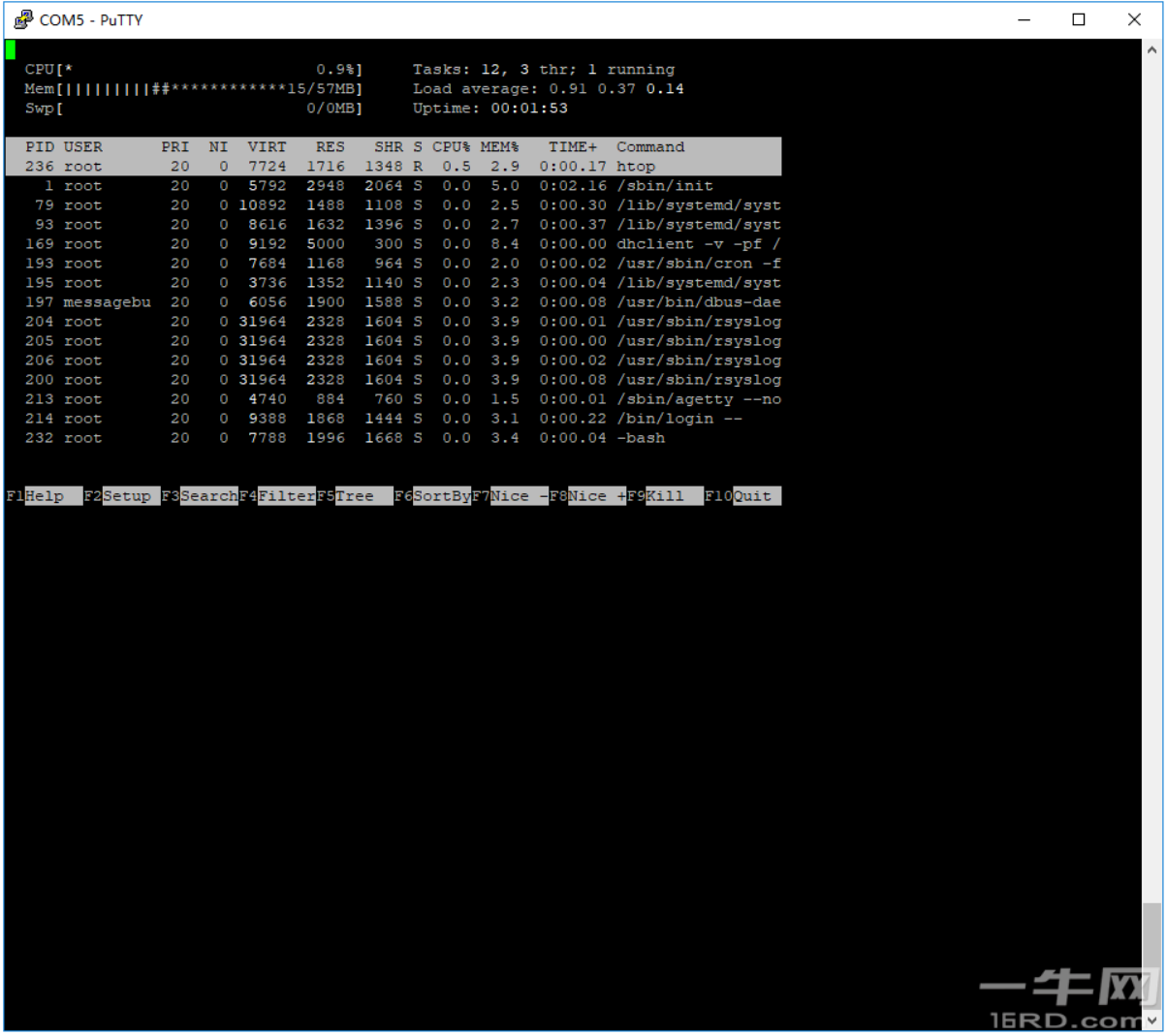
在编辑框的末尾添加以下命令，然后保存退出并重启目标板：

```
01.  /opt/swap/swapfile swap swap defaults 0 0
```
复制代码

重启目标板后在启动过程中出现了如下错误：




登录系统后使用htop命令，可以看到SWAP交换区大小仍然为0。

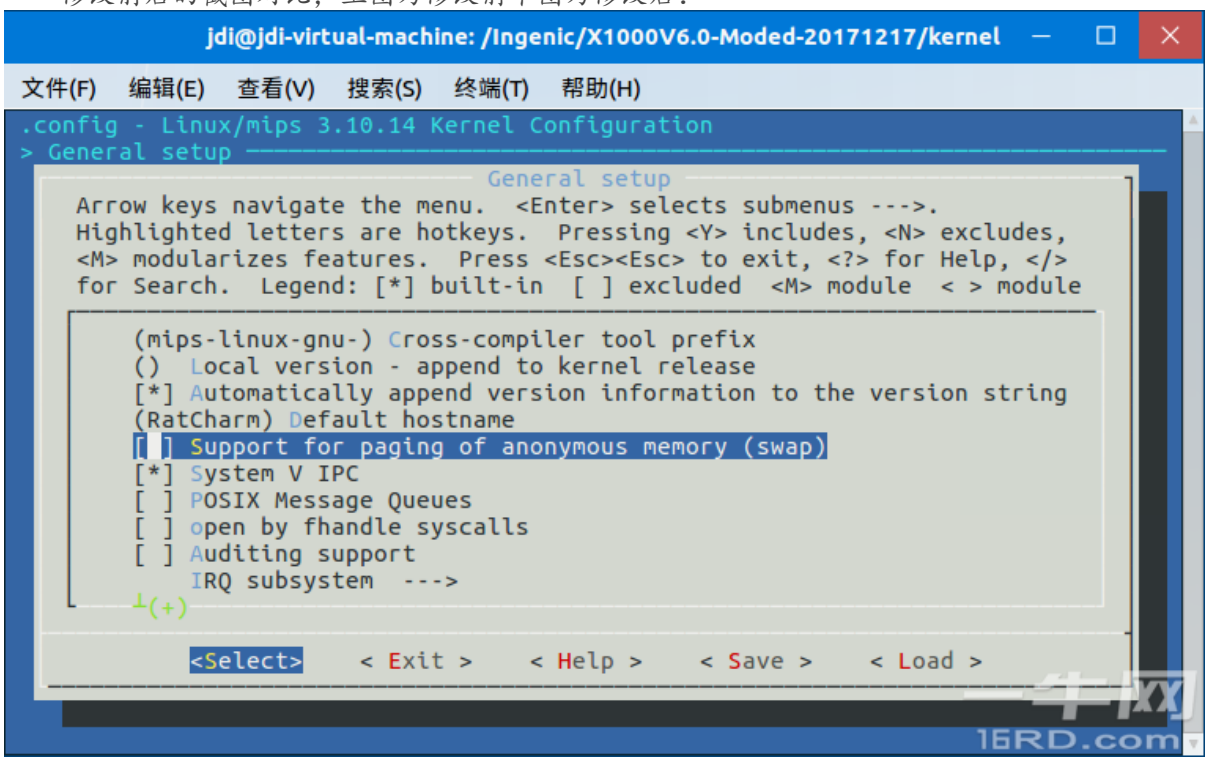


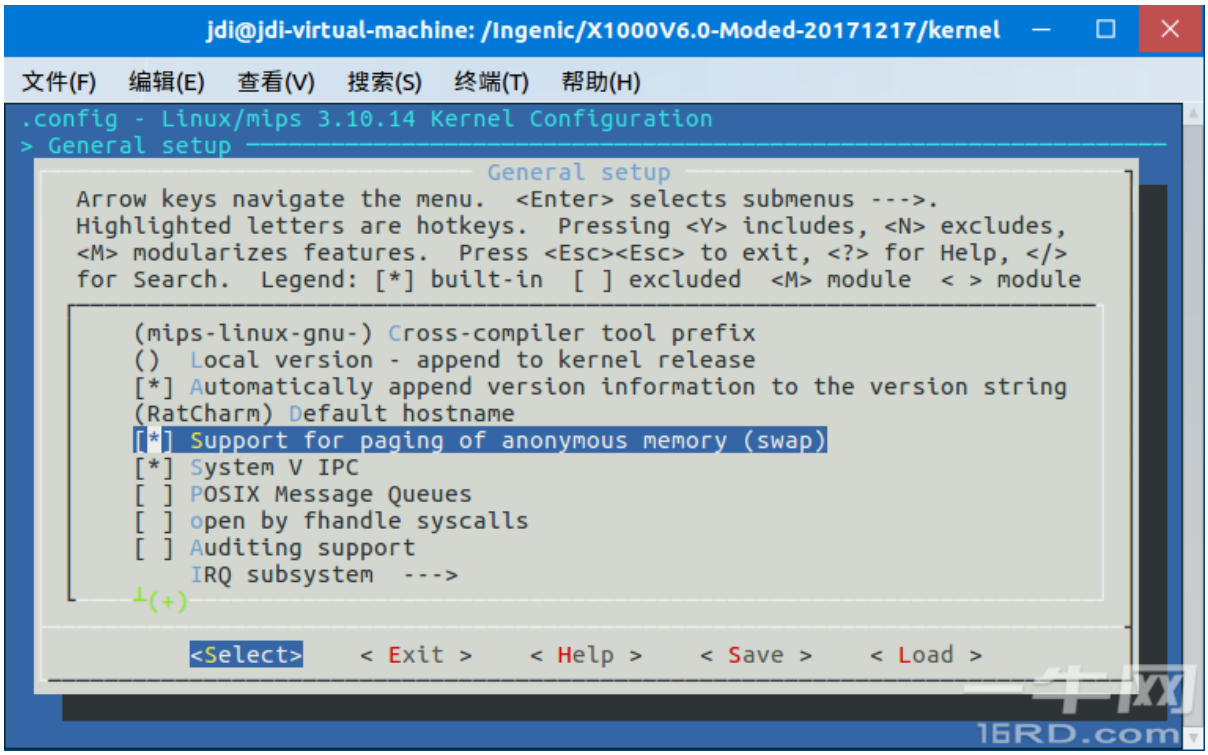

分析原因后判断是kernel内没有配置对swap的支持，需进行如下配置：

```
01.     | Symbol: SWAP [=y]                                                    |
02.     | Type  : boolean                                                      |
03.     | Prompt: Support for paging of anonymous memory (swap)                |
04.     |   Location:                                                          |
05.     |     -> General setup                                                 |
06.     |   Defined at init/Kconfig:182                                        |
07.     |   Depends on: MMU [=y] && BLOCK [=y]                                 |
```

复制代码

修改前后的截图对比，上图为修改前下图为修改后：

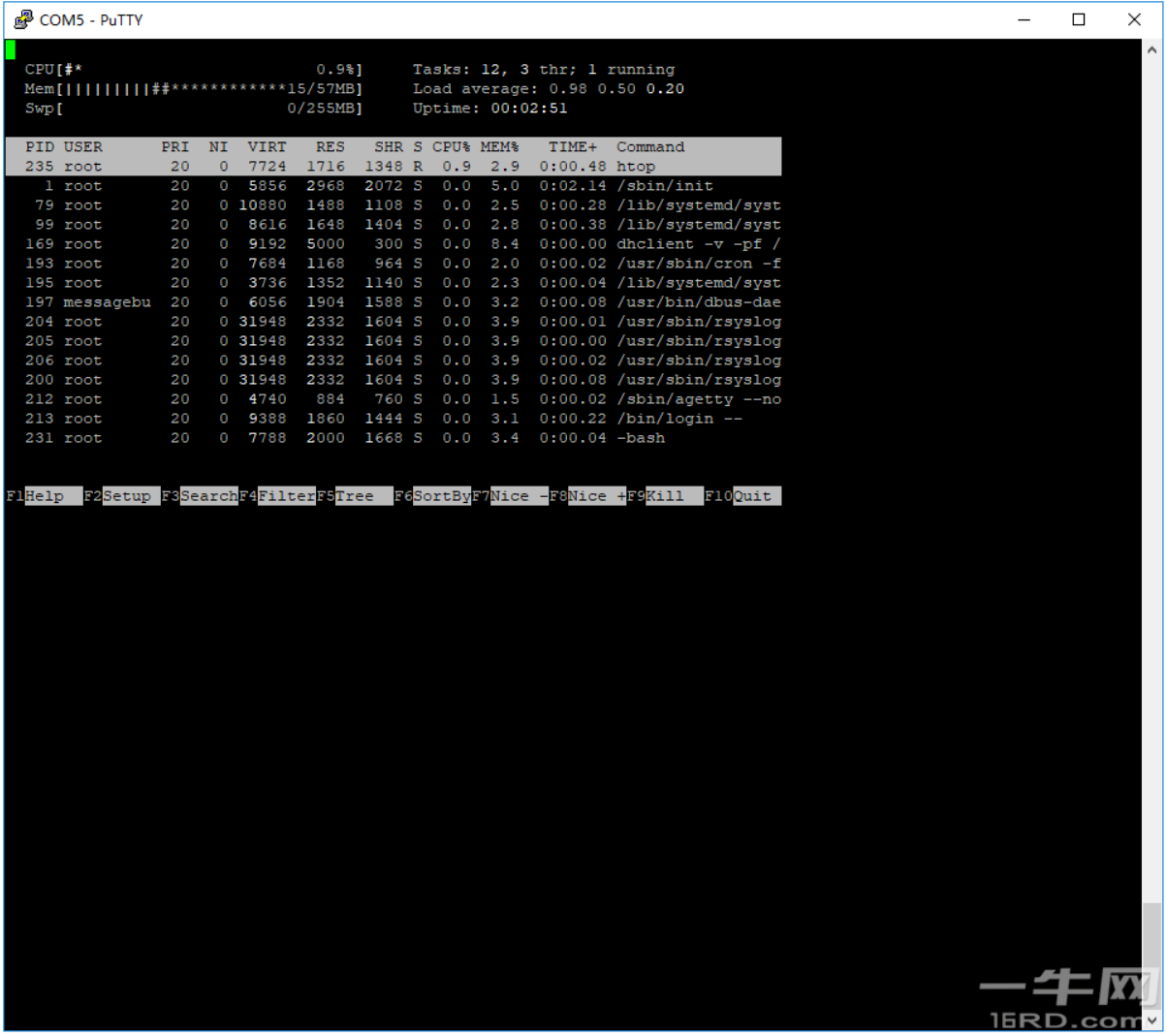



重新编译好kernel并将其烧写到目标板，准备再次进行引导。

- 第六次尝试：

SWAP交换区挂载出错的问题已经解决，登录系统后使用htop命令，
可见SWAP已经正常挂载，总容量为255MB，当前使用量为0。




以上设置SWAP分区的操作也可以放到打包rootfs的环节进行，在使用exit退出打包脚本前，
使用同样的方法逐个完成前面的各项操作，然后使用exit退出脚本并制作ext4镜像即可。
目前为止，Debian8的基本功能都已经可以正常运行，剩余的就是一些零碎的善后工作。

## 最后，还有一些杂七杂八的善后工作，列了一个清单：

- **添加普通用户：**不少操作不适合在root账户下进行，因此需要建立一个普通账户。
- **设置sudo相关配置：**在普通用户下很多操作需要sudo来临时提升权限，因此需要进行相关配置。
- **安装常用的软件包：**安装日常使用中经常会用到的一些软件。

接下来就来逐步完善以上内容，首先是添加普通用户，使用串口登录root账户，然后执行以下命令：

```
01. adduser rat
```

复制代码

然后根据提示输入登录密码、个人信息等(可不填)，顺利完成后，
就会建立起一个名为rat的普通账户。然后执行以下命令：

```
01. su rat
02. nano /home/rat/.bashrc
```

复制代码

然后在编辑框末尾添加以下内容：

```
01. export PATH=$PATH:/sbin
```

复制代码

保存并退出，此时我们就完成了对rat账户的所有配置工作，下次登录时rat账户就已经处于可用状态。

配置完rat这个普通账户后，自然就是配置sudo这个在普通账户中经常会使用到的命令。
配置过程仍然需要在root账户下进行，登录root账户后依次执行以下命令：

```
01. apt-get install sudo
02. chown root:root /usr/bin/sudo
```

```
chmod 4755 /usr/bin/sudo
```
复制代码

接下来还需要对/etc/sudoers和/etc/hosts两个文件进行修改。首先修改/etc/sudoers，执行以下命令：

```
vim /etc/sudoers
```
复制代码

然后在编辑框中进行修改，以下是原始代码：

```
# User privilege specification
root    ALL=(ALL:ALL) ALL
```
复制代码

修改后的代码如下：

```
# User privilege specification
root        ALL=(ALL:ALL) ALL
rat         ALL=(ALL:ALL) ALL
```
复制代码

完成以上修改后对文件进行保存。需要注意的是因为是只读文件，所以保存时需要使用wq！命令强制保存。
紧接着再来对/etc/hosts进行修改，执行以下命令：

```
vim /etc/hosts
```
复制代码

在编辑框中进行修改，以下是原始代码

```
127.0.0.1       localhost
::1             localhost ip6-localhost ip6-loopback
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters
```
复制代码

修改后的代码如下：

```
127.0.0.1       localhost debian
::1             localhost ip6-localhost ip6-loopback
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters
```
复制代码

完成修改后保存并退出编辑框。至此，对sudo的所有配置均已完成。

既然运行了Debian8，那么自然要利用好它的软件生态，可以在root账户下使用以下命令安装常用软件包：

```
apt-get install ssh build-essential gfortran openjdk-7-jdk
```
复制代码

完成安装后，可以在目标板对一些简单的程序进行本地编译(复杂的程序也可编译，但耗时会很长)，
并且可以通过网络使用ssh登录到目标板进行操作或使用scp传输文件，增加了灵活性。
以上所有善后工作也可以放到打包rootfs的环节进行，在使用exit退出打包脚本前，
使用同样的方法逐个完成以上各项操作，然后使用exit退出脚本并制作ext4镜像，
如此，得到的就是配置好了普通用户、sudo、常用软件包的镜像文件。

- **附上修改好的打包rootfs所需的脚本，以及最终完成的kernel镜像和rootfs镜像：**

已修改好的用于打包Debian8 rootfs的脚本：


mkdebianrfs.zip (2.65 KB, 下载次数: 1 )

最终版本的u-boot、kernel和rootfs镜像：

u-boot-with-spl-mbr-gpt.zip (95.18 KB, 下载次数: 2 )

uImage.zip (2.58 MB, 下载次数: 2 )

debian8-rootfs.zip (309.78 MB, 下载次数: 2 )