



浙江工业大学

# 实验报告

课程：嵌入式系统 A

第四次实验

姓 名 凌智城

学 号 201806061211

专业班级 通信工程 1803 班

老 师 黄国兴

学 院 信息工程学院

提交日期 2021 年 5 月 26 日

## 实验 12：mount 挂载实验

重新提交第三次实验的挂载实验，上次实验后半部分没做完

### 一、实验目的

1. 掌握配置 NFS 服务的方法。
2. 掌握 mount 挂在 usb/sd 的方法。

### 二、实验内容

1. 配置 NFS 服务。
2. mount 挂在 usb/sd 设备。

### 三、实验步骤

#### 步骤 1：连接设备

打开 PC 机上的串口调试工具，开启实验箱上的电源，按空格键进入实验箱上的板上 Linux 系统。

若是连接实验室的三台服务器需要先 SSH 连接上服务器，若使用本地虚拟机则打开虚拟机 bash 即可。

接着查看串口号，通过 putty 软件使用串口通信方式连接实验箱，如图 12-1 所示所示：选择 putty 串口连接试验箱如图 12-2 所示：

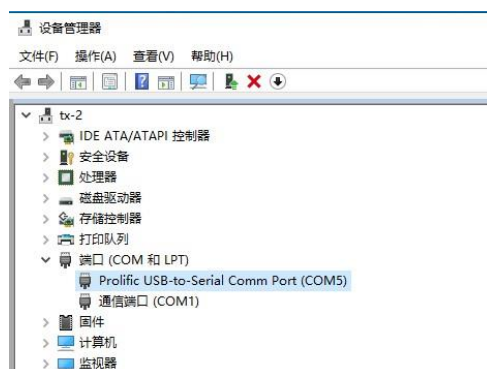


图 12-1 查看串口端号

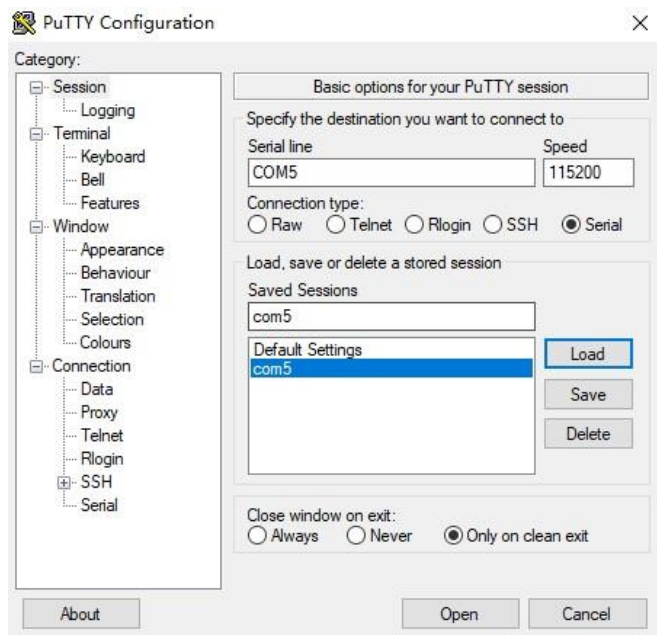


图 12-2 putty 串口连接配置

输入启动参数，接着启动内核，如图 12-3 所示：

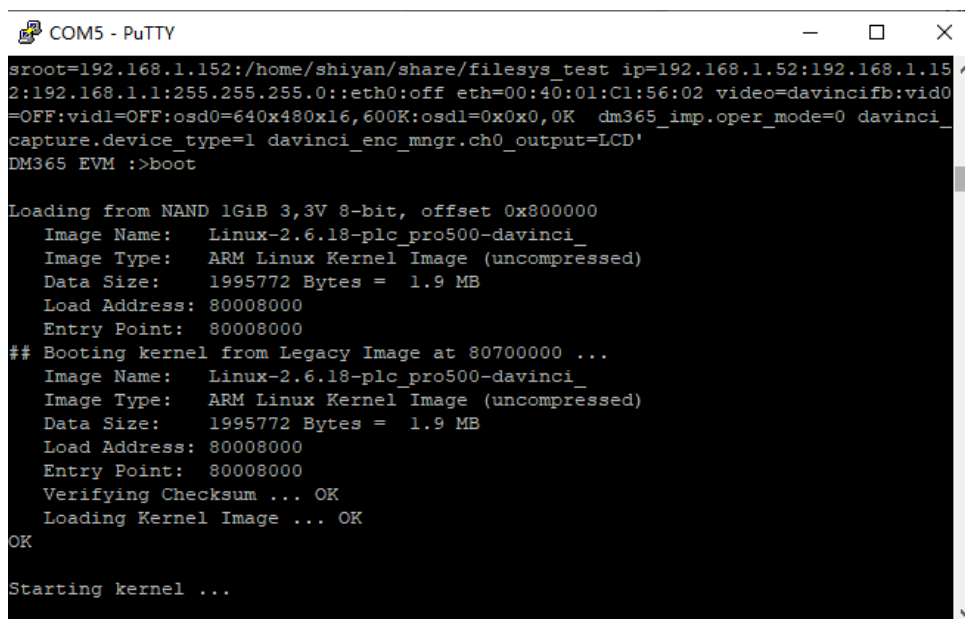


图 12-3 输入启动参数

```
setenv    bootargs    'mem=110M    console=ttyS0,115200n8    root=/dev/nfs    rw
nfsroot=192.168.1.152:/home/shiyan/share/filesys_test
ip=192.168.1.52:192.168.1.152:192.168.1.1:255.255.255.0::eth0:off  eth=00:40:01:C1:56:02
video=davincifb:vid0=OFF:vid1=OFF:osd0=640x480x16,600K:osd1=0x0x0,0K
dm365_imp.oper_mode=0                                davinci_capture.device_type=1
davinci_enc_mgr.ch0_output=LCD'
```

使用实验室服务器的和使用虚拟机的配置有所不同，使用虚拟机的先用 `ifconfig` 命令在 `inet addr` 处查看 IP，根据试验箱的 MAC 地址更改数据。

输入用户名 `root` 登录试验箱如图 12-4 所示

```
root

Welcome to MontaVista(R) Linux(R) Professional Edition 5.0.0 (0801921).

login[753]: root login on 'console'
/*****Set QT environment*****/
[root@zjut ~]#
```

图 12-4 登录试验箱

**步骤 2：**配置 nfs 服务器设置（使用服务器用 `putty` 连接服务器，使用虚拟机则用 VMware 进入 Ubuntu 系统）

1. 进入 Linux 服务器系统的 `/etc` 目录，命令如下：

```
shiyang@ubuntu:/$ cd /etc/
```

```
shiyang@ubuntu:/etc$
```

2. 编辑 `/etc/exports` 的文件，`sudo` 命令是进入 `root` 权限，这里需要输入登录密码，命令如下：

```
shiyang@ubuntu:/etc$ sudo vi exports
```

```
[sudo] password for shiyang:
```

进入如下所示 `exports` 文件，再 `exports` 文件中添加一行：

```
/home/挂载目录 192.168.*.*(rw, sync, no_root_squash)
```

（即在文件底部添加，保存退出，192.168.\*.\*根据服务器 IP 和网关确定）

至此已完成 NFS 服务器配置，接下来启动 NFS 服务：

启动 NFS 的命令如下：

```
$ sudo ./etc/rc.d/init.d/nfs start
```

如果之前已启动 NFS，更改后可用以下命令：

```
$ sudo /etc/init.d/nfs-kernel-server restart
```

```
shiyang@ubuntu: /etc
shiyang@ubuntu:~$ cd /etc/
shiyang@ubuntu:/etc$ sudo vi exports
[sudo] password for shiyang:

[1]+  Stopped                  sudo vi exports
shiyang@ubuntu:/etc$ sudo /etc/init.d/nfs-kernel-server start
* Exporting directories for NFS kernel daemon...      [ OK ]
* Starting NFS kernel daemon                          [ OK ]
shiyang@ubuntu:/etc$
```

图 12-5 启动 NFS

### 步骤 3：文件夹挂载

#### 1. 挂载

服务器端的 NFS 服务配置完成以后启动实验板，在串口调试工具中开始挂在文件夹，在 mount 之前必须先配置。加上 `ifconfig eth0 192.168.1.***` 命令，修改实验板 IP。（上述 IP 为实验箱的具体 IP，注意要和被挂载的服务器处在同一网段。）

mount 过程如下：（实验箱上进行）

```
[root]# mount -t nfs -o nolock 192.168.1.152:/home/shiyang/share/filesys_test
/mnt/mtd/

[root]#
```

验证挂载是否成功，输入 df 命令查看，结果增加一行出现：

```
[root]#df
```

Filesystem	1K-blocks	Used	Availabed	Use%	Mounted on
192.168.1.***	:	/home/shiyang/share/	193241632	102773502	80652000
56%		/mnt/mtd			

从上可以看出已经将服务器上的 `/home/shiyang/share/nfs`（`192.168.1.***:/home/shiyang/shiare/nfs`）目录挂载到了实验箱文件系统的 `/mnt/mtd` 目录下。也就是说此时实验箱可以通过 `/mnt/mtd` 目录直接访问服务器上的 `/home/shiyang/share/nfs` 目录。可以在服务器端进入 `/home/shiyang/share/nfs` 目录和在实验箱中进入 `/mnt/mtd` 目录对比里面的内容，可以发现内容是一样的，并且在任意端向目录中创建新文件，在另一端均可见。

```
login[748]: root login on 'console'
/*****Set QT environment*****/
< -o nolock 192.168.1.152:/home/shiyan/share/filesys_test /mnt/mtd/
[root@zjut ~]# df
Filesystem            1k-blocks      Used Available Use% Mounted on
tmpfs                  53868         92     53776   0% /tmp
tmpfs                  10240         32     10208   0% /dev
tmpfs                  53868          0     53868   0% /dev/shm
192.168.1.152:/home/shiyan/share/filesys_test 43218944 10396672 30604288 25%
/mnt/mtd
[root@zjut ~]#
```

图 12-6 文件夹挂载成功

## 2. 卸载

为了将/192.168.1.\*\*\*:/home/shiyan/shiare/nfs 目录与/mnt/mtd 目录卸载分开, 首先退到 root 目录下 (cd / 请注意卸载命令发生在实验箱端, 且一定要在卸载挂载前退出挂载目录, 否则会报错, 报错内容为设备忙), 需要使用 umount 命令(umount 被挂载目录), 如下所示:

```
[root]# umount /mnt/mtd
```

df 查看后已无显示服务器的内容, 完成卸载。

```
cd /
[root@zjut /]# umount /mnt/mtd
[root@zjut /]# df
Filesystem            1k-blocks      Used Available Use% Mounted on
tmpfs                  53868        112     53756   0% /tmp
tmpfs                  10240         32     10208   0% /dev
tmpfs                  53868          0     53868   0% /dev/shm
[root@zjut /]#
```

图 12-7 文件夹卸载成功

## 步骤 4: usb 挂载 (FAT32 格式 U 盘)

1. 将 U 盘插入实验板的 USB 接口处, 实验板中的串口调试工具出现以下信息提示:

```
[root]# [149.340000] usb 1-1.3:new high speed USB device using musb_hdrc and
address 4
```

```
[root@zjut /]# [ 312.740000] usb 1-1.3: new high speed USB device using musb_hdr
c and address 6
```

图 12-8 插入 U 盘

2. 使用 fdisk-l 查看盘符详细, 如下所示:

```
[root]# fdisk -l
```

```
Disk /dev/sda: 4057 MB, 4057989120 bytes
```

```
91 heads, 45 sectors/track, 1935 cylinders
```

```
Units = cylinders of 4095 * 512 = 2096640 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1		1	1936	3962852	b	Win95 FAT32

```
[root]#
```

```
[root@zjut /]# fdisk -l

Disk /dev/sda: 128.3 GB, 128320801792 bytes
255 heads, 63 sectors/track, 15600 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1            1         15601     125313203    7  HPFS/NTFS
```

图 12-9 输入 fdisk -l 查看盘符

3. 创建一个/mnt/usb 文件夹，如下所示：

```
[root]# mkdir /mnt/usb
```

```
[root]#
```

4. 把 sda1 盘符 mount 到/mnt/usb 文件夹上，如下所示：

```
[root]# mount/dev/sda1/mnt/usb/
```

```
[root]#
```

```
[root@zjut /]# mount /dev/sda1 /mnt/usb/
[ 997.730000] yaffs: dev is 8388609 name is "sda1"
[ 997.730000] yaffs: passed flags ""
[ 997.740000] yaffs: Attempting MTD mount on 8.1, "sda1"
[ 997.770000] yaffs: dev is 8388609 name is "sda1"
[ 997.770000] yaffs: passed flags ""
[ 997.780000] yaffs: Attempting MTD mount on 8.1, "sda1"
mount: mounting /dev/sda1 on /mnt/usb/ failed: Invalid argument
[root@zjut /]#
```

图 12-10 尝试挂载 U 盘

但是因为插入 U 盘格式为 exFAT，在实验箱中未能成功挂载

5. 进入/mnt/usb/文件夹，查看文件夹中的内容，如下所示：

```
[root]# cd /mnt/usb/
```

```
[root]# ls
```

h264	bin	disk.tar.gz	etc	data.h264	dev	init	lib
linuxrc	mnt	proc	root	sbin	share	shm	sys

```
[root]#
```

6. 卸载 U 盘，先退到根目录下，再解除挂载，如下所示：

```
[root@zjut usb]# cd /
```

```
[root@zjut ~]# umount /mnt/usb
```

7. 解除挂在以后，可再次进入/mnt/usb/文件夹，输入 ls 查看，若文件夹内已经没有任何内容，说明解除挂在成功：

```
[root@zjut ~]# cd /mnt/usb
```

```
[root@zjut usb]# ls
```

```
[root@zjut usb]#
```

## 四、心得与体会

在挂载实验中部分同学使用的是实验室的三台服务器，部分同学使用的是 VMware 创建的 Ubuntu 虚拟机，IP 等配置有所不同，并且由于没分配好用户和 IP 导致出现了较多的冲突；同时实验箱卡顿也比较严重，常常出现死机的状况，重启后并不能进入实验箱系统，最后并没能在实验室中做全部实验，总结出两个问题就是课前没有及时预习，对挂载实验的操作步骤已经 IP 分配并不熟悉，反复在同一步骤上浪费过多时间，第二个就是自主学习能力不够强，对出现的问题不能及时找到解决方法已知拖到了最后还没完成。但经过这次挂载实验，对如何加载文件系统和卸载有了更加清楚的认识，同时在之后的实验中也将更加重视实验的预习。



## 实验五：Linux 交叉编译环境

### 一、 实验目的

1. 理解交叉编译的原理和概念。
2. 掌握在 Linux 下搭建交叉编译平台的方法。
3. 掌握使用交叉编译平台编译源代码。

### 二、 实验内容

1. 正确运行实验箱。
2. 通过串口线将实验箱和 PC 机链接。
3. 在 PC 机的 Linux 操作系统上搭建交叉编译平台，并编译程序。
4. 在实验箱上运行交叉编译程序结果。

### 三、 实验步骤

**步骤 1：**直接进入虚拟机终端。（若是服务器择偶那个 putty 进入服务器）

**步骤 2：**搭建交叉编译环境。

创建一个文件夹 mv\_pro\_5.0，进入文件夹 mv\_pro\_5.0，将/home/shiyan/2021（虚拟机是在/home/shiyan 目录）目录下的软件包 mvtools5\_0\_0801921\_update.tar 复制到当前目录 mv\_pro\_5.0 下（注意不能省略最后一条语句中的“.”，且前面有空格）：

```
# mkdir mv_pro_5.0
```

```
# cd mv_pro_5.0
```

解压缩 mvtools5\_0\_0801921\_update.tar 软件包，解压缩后出现 montavista 文件夹：

```
# tar zxvf mvtools5_0_0801921_update.tar.gz
```

```
root@ubuntu: /mv_pro_5.0
montavista/common/share/misc/
montavista/common/share/misc/bison/
montavista/common/share/misc/bison/lalr1.cc
montavista/common/share/misc/bison/README
montavista/common/share/misc/bison/c.m4
montavista/common/share/misc/bison/blr.c
montavista/common/share/misc/bison/m4sugar/
montavista/common/share/misc/bison/m4sugar/m4sugar.m4
montavista/common/share/misc/bison/yacc.c
montavista/common/share/apt/
montavista/common/share/apt/scripts/
montavista/common/share/apt/scripts/downloadsrc.lua
montavista/common/share/apt/scripts/list-updates.lua
montavista/common/share/apt/scripts/list-upgraded.lua
montavista/common/var/
montavista/common/var/cache/
montavista/common/var/cache/apt/
montavista/common/var/cache/apt/archives/
montavista/common/var/cache/apt/archives/partial/
montavista/common/var/lib/
montavista/common/var/lib/apt/
montavista/common/var/lib/apt/lists/
montavista/common/var/lib/apt/lists/partial/
root@ubuntu: /mv_pro_5.0#
```

图 5-1 创建文件夹拷贝压缩包并解压

配置系统环境变量，把交叉编译工具链路径添加到环境变量 PATH 中去，使其可以在任何目录下使用，进入/etc/profile 文档：

```
# vim /etc/profile
```

点击插入键 i，在文件最后一行添加：

服务器：

```
export PATH=$PATH:/home/stX/mv_pro_5.0/montavista/pro/devkit/arm/v5t_le/bin
```

虚拟机：

```
export PATH=$PATH:/home/mv_pro_5.0/montavista/pro/devkit/arm/v5t_le/bin
```

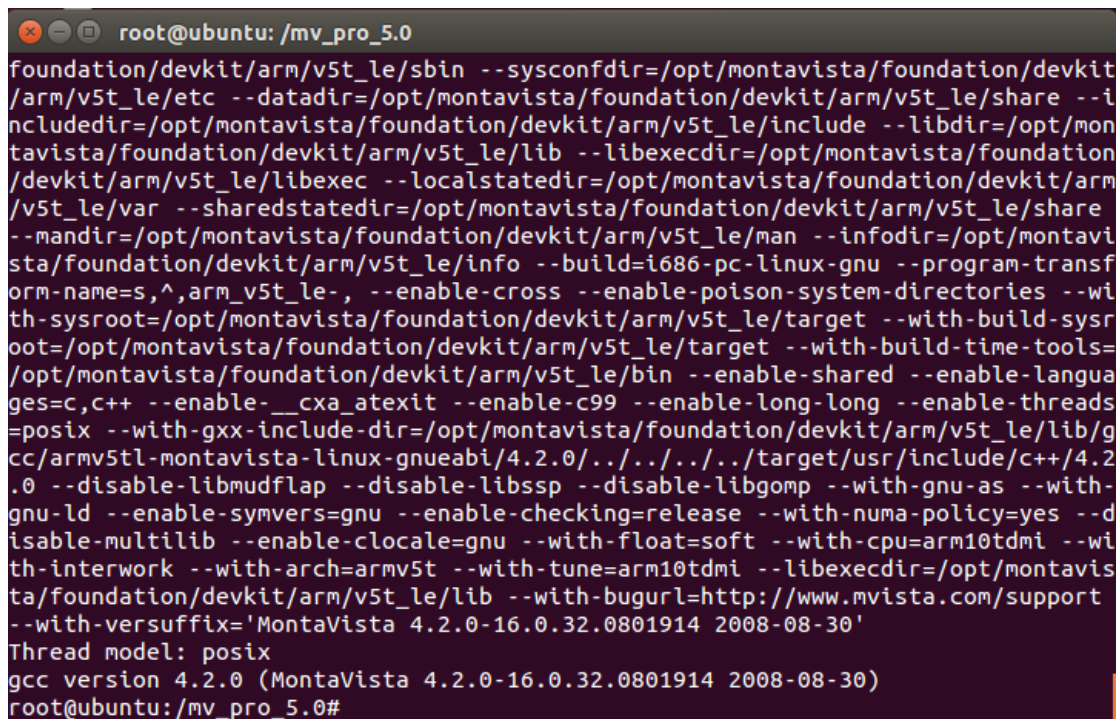
```
root@ubuntu: /mv_pro_5.0
if [ "${id -u}" -eq 0 ]; then
    PS1='# '
else
    PS1='$ '
fi
fi
# The default umask is now handled by pam_umask.
# See pam_umask(8) and /etc/login.defs.
if [ -d /etc/profile.d ]; then
    for i in /etc/profile.d/*.sh; do
        if [ -r $i ]; then
            . $i
        fi
    done
    unset i
fi
export PATH=$PATH:/home/shiyan/montavista/pro/devkit/arm/v5t_le/bin
export PATH=$PATH:/home/mv_pro_5.0/montavista/pro/devkit/arm/v5t_le/bin
```

图 5-2 配置系统环境变量

点击 `ecs` 键退出输入，输入:`wq!`推出文档。使环境变量生效：

```
# source /etc/profile
```

检查交叉编译环境是否搭建成功：在命令行输入 `arm_v5t_le-gcc-v`，打印出版本信息，表示交叉编译环境搭建成功。



```
root@ubuntu: /mv_pro_5.0
foundation/devkit/arm/v5t_le/sbin --sysconfdir=/opt/montavista/foundation/devkit/arm/v5t_le/etc --datadir=/opt/montavista/foundation/devkit/arm/v5t_le/share --includedir=/opt/montavista/foundation/devkit/arm/v5t_le/include --libdir=/opt/montavista/foundation/devkit/arm/v5t_le/lib --libexecdir=/opt/montavista/foundation/devkit/arm/v5t_le/libexec --localstatedir=/opt/montavista/foundation/devkit/arm/v5t_le/var --sharedstatedir=/opt/montavista/foundation/devkit/arm/v5t_le/share --mandir=/opt/montavista/foundation/devkit/arm/v5t_le/man --infodir=/opt/montavista/foundation/devkit/arm/v5t_le/info --build=i686-pc-linux-gnu --program-transform-name=s,^,arm_v5t_le-, --enable-cross --enable-poison-system-directories --with-sysroot=/opt/montavista/foundation/devkit/arm/v5t_le/target --with-build-sysroot=/opt/montavista/foundation/devkit/arm/v5t_le/target --with-build-time-tools=/opt/montavista/foundation/devkit/arm/v5t_le/bin --enable-shared --enable-languages=c,c++ --enable-__cxa_atexit --enable-c99 --enable-long-long --enable-threads --enable-posix --with-gxx-include-dir=/opt/montavista/foundation/devkit/arm/v5t_le/lib/gcc/armv5t_le-montavista-linux-gnueabi/4.2.0/../../../../target/usr/include/c++/4.2.0 --disable-libmudflap --disable-libssp --disable-libgomp --with-gnu-as --with-gnu-ld --enable-symvers=gnu --enable-checking=release --with-numa-policy=yes --disable-multilib --enable-clocale=gnu --with-float=soft --with-cpu=arm10tdmi --with-interwork --with-arch=armv5t --with-tune=arm10tdmi --libexecdir=/opt/montavista/foundation/devkit/arm/v5t_le/lib --with-bugurl=http://www.mvista.com/support --with-versuffix='MontaVista 4.2.0-16.0.32.0801914 2008-08-30'
Thread model: posix
gcc version 4.2.0 (MontaVista 4.2.0-16.0.32.0801914 2008-08-30)
root@ubuntu: /mv_pro_5.0#
```

图 5-3 交叉编译环境检测

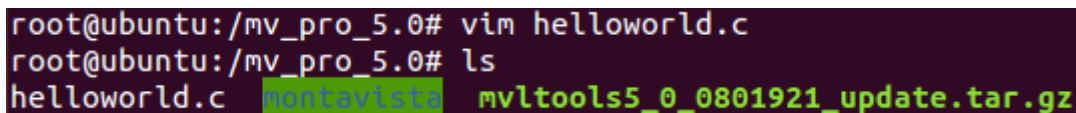
### 步骤 3：小程序测试

- 1) 步骤2中搭建了交叉编译环境，接下来交叉编译一个小程序“helloworld”，挂载到实验箱上，新建 vim 文件（`vim helloworld.c`），输入以下内容：

```
#include<stdio.h>

int main()
{
printf("hello world !\n");
return 0;
}
```

点击 `esc` 键退出输入，输入:`wq!`退出文档。



```
root@ubuntu: /mv_pro_5.0# vim helloworld.c
root@ubuntu: /mv_pro_5.0# ls
helloworld.c  montavista  mvltools5_0_0801921_update.tar.gz
```

图 5-4 创建 hello world 程序

## 2) 交叉编译

生成二进制可执行文件 `helloworld`，其中 `helloworld.c` 为交叉编译的程序，`-o` 表示输出，`helloworld` 表示生成的二进制可执行文件名：

```
# arm_v5t_le-gcc helloworld.c -o helloworld  
root@ubuntu:/mv_pro_5.0# arm_v5t_le-gcc helloworld.c -o helloworld  
root@ubuntu:/mv_pro_5.0# ls  
helloworld helloworld.c montavista mvltools5_0_0801921_update.tar.gz  
root@ubuntu:/mv_pro_5.0#
```

图 5-5 生成二进制可执行文件

在 PC 上运行生成的二进制可执行文件 `helloworld`：

```
# ./helloworld
```

提示 `-bash:./helloworld:cannot execute binary file: Exec format error`，即不能执行该二进制可执行文件。

```
root@ubuntu:/mv_pro_5.0# ./helloworld  
bash: ./helloworld: cannot execute binary file: Exec format error  
root@ubuntu:/mv_pro_5.0#
```

图 5-6 报错不能执行可执行文件

## 3) 正确连接实验箱和 PC 机

将 PC 机与开发板通过 USB 转串口线正确连接，将开发板的电源线、网线正确连接，插上电源。

## 4) 登录 Putty 的 COM 口端

打开设备开关，在 PuTTY 的 COM 口端进行操作，当实验板有打印消息时，按 `enter` 键使系统停止启动，输入启动参数：

保存启动参数# `saveenv`

## 5) 通过实验箱运行交叉编译生成的可执行文件

在 PuTTY 的服务器端进行操作，将生成的二进制可执行文件 `helloworld` 由它所在的目录复制到文件系统所在目录的 `/opt/dm365` 下(eg:服务器文件系统所在目录为 `/home/shiyan/filesys_test`，就复制到 `/home/stX/filesys_test/opt/dm365` 目录)；(虚拟机文件系统所在目录为 `/home/shiyan/share/filesys_test/`，就复制到 `/home/shiyan/share/filesys_test/opt/dm365` 目录)：

服务器：# `sudo cp helloworld /home/stX/filesys_test/opt/dm365`

虚拟机：# `sudo cp helloworld /home/shiyan/share/filesys_test/opt/dm365`

```
root@ubuntu:/# cd mv_pro_5.0
root@ubuntu:/mv_pro_5.0# ls
helloworld helloworld.c montavista mvltools5_0_0801921_update.tar.gz
root@ubuntu:/mv_pro_5.0# sudo cp helloworld /home/shiyan/share/filesys_test/opt/dm365
root@ubuntu:/mv_pro_5.0#
```

图 5-7 将二进制可执行文件复制到文件系统的/opt/dm365 下

在 putty 端口操作：

进入可执行文件所在目录/opt/dm365

```
# cd /opt/dm365
```

运行二进制可执行文件 helloworld:

```
# helloworld
```

显示结果如下：

```
[root@zjut ~]# cd /opt/dm365
[root@zjut dm365]# helloworld
hello world!
[root@zjut dm365]#
```

图 5-8 交叉编译成功

## 四、心得与体会

通过这次交叉编译实验，学会了如何搭建交叉编译环境，有了实验四挂载实验的基础后，实验进行地较为顺利，出现的问题也能够独立解决，对后续实验的顺利进行奠定了良好的基础。

## 实验六：Linux 内核编译实验

### 一、 实验目的

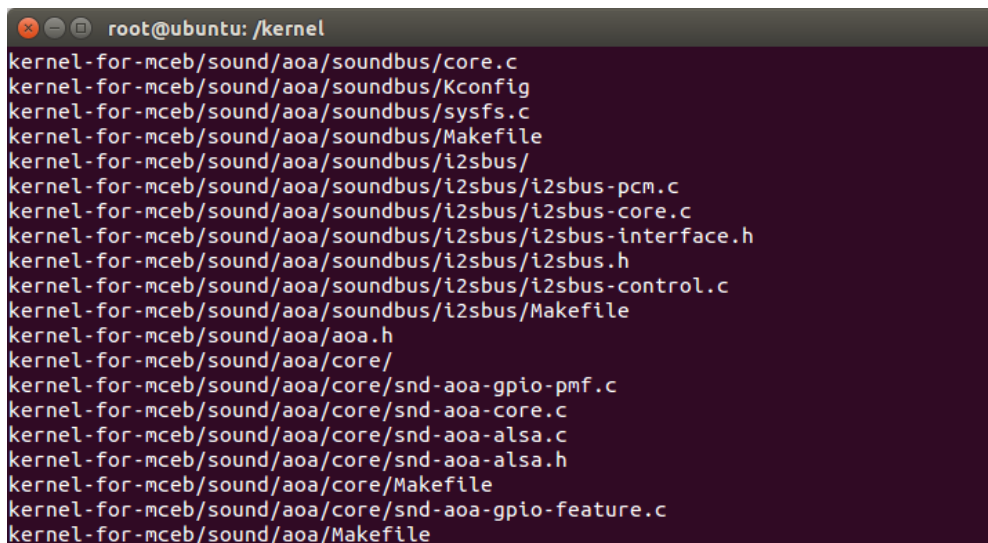
1. 掌握配置和编译 Linux 内核的方法。
2. 掌握 Linux 内核的编译过程。
3. 熟悉 Linux 系统一些基本内核配置。
4. 熟悉熟悉内核编译的常见命令。

### 二、 实验内容

1. 利用编译进内核的方法配置内核。
2. 利用动态加载方法配置内核。
3. 编译 Linux 内核镜像。
4. 编译 Linux 内核模块。

### 三、 实验步骤

**步骤 1:** 登录虚拟机找到内核目录，移动物联网实验箱的内核基于 2.6.18 该进来的 kernel-for-mecb。



```
root@ubuntu: /kernel
kernel-for-mceb/sound/aoa/soundbus/core.c
kernel-for-mceb/sound/aoa/soundbus/Kconfig
kernel-for-mceb/sound/aoa/soundbus/sysfs.c
kernel-for-mceb/sound/aoa/soundbus/Makefile
kernel-for-mceb/sound/aoa/soundbus/i2sbus/
kernel-for-mceb/sound/aoa/soundbus/i2sbus/i2sbus-pcm.c
kernel-for-mceb/sound/aoa/soundbus/i2sbus/i2sbus-core.c
kernel-for-mceb/sound/aoa/soundbus/i2sbus/i2sbus-interface.h
kernel-for-mceb/sound/aoa/soundbus/i2sbus/i2sbus.h
kernel-for-mceb/sound/aoa/soundbus/i2sbus/i2sbus-control.c
kernel-for-mceb/sound/aoa/soundbus/i2sbus/Makefile
kernel-for-mceb/sound/aoa/aoa.h
kernel-for-mceb/sound/aoa/core/
kernel-for-mceb/sound/aoa/core/snd-aoa-gpio-pmf.c
kernel-for-mceb/sound/aoa/core/snd-aoa-core.c
kernel-for-mceb/sound/aoa/core/snd-aoa-alsa.c
kernel-for-mceb/sound/aoa/core/snd-aoa-alsa.h
kernel-for-mceb/sound/aoa/core/Makefile
kernel-for-mceb/sound/aoa/core/snd-aoa-gpio-feature.c
kernel-for-mceb/sound/aoa/Makefile
```

图 6-1 创建文件夹 kernel 复制进压缩包后解压

服务器: #cp /home/shiyan/2021/ kernel-for-mceb.tar.gz ./

虚拟机: #cp /home/shiyan/Desktop/shiyan/ kernel-for-mceb.tar.gz ./

## 步骤 2: 进入内核进行配置

在配置前首先输入 `sudo -s`, 并输入密码, 登录超级用户, 之后再次输入命令 `vim/etc/profile` 检查交叉编译路径是否正确。之后退出, 输入命令 `source /etc/profile`, 使环境变量生效。

进入内核目录, 执行命令为 `cd kernel-for-mceb`。如果不是第一次编译内核, 先运行 `make mrproper` 清除以前的配置, `huidaomoren` 配置。然后继续进行内核配置, 执行命令为 `make menuconfig`。出现窗口如下图所示:

```
root@ubuntu:/kernel# cd kernel-for-mceb
root@ubuntu:/kernel/kernel-for-mceb# make mrproper
CLEAN arch/arm/boot/compressed
CLEAN arch/arm/boot
CLEAN /kernel/kernel-for-mceb
CLEAN arch/arm/kernel
CLEAN drivers/char
CLEAN drivers/video/logo
CLEAN init
CLEAN kernel
CLEAN lib
CLEAN usr
CLEAN .tmp_versions
CLEAN include/asm-arm/mach-types.h include/asm-arm/arch include/asm-arm/.arc
h vmlinux System.map .tmp_kallsyms1.o .tmp_kallsyms1.S .tmp_kallsyms2.o .tmp_kal
lsyms2.S .tmp_vmlinux1 .tmp_vmlinux2 .tmp_System.map
rm: cannot remove 'include/asm-arm/arch': Is a directory
make: *** [clean] Error 1
root@ubuntu:/kernel/kernel-for-mceb#
```

图 6-2 回到默认配置

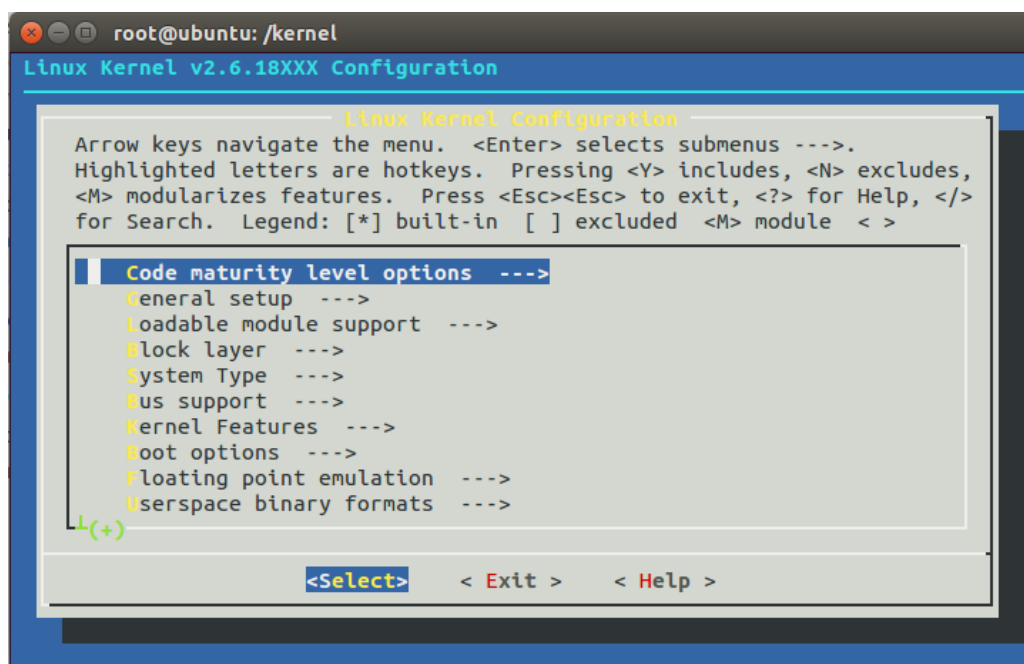


图 6-3 配置内核

编译内核时往往根据自己的需要来编译自己所需要的。下面是一些常见的驱动选项。

选择	说明
MemoryTechnologyDevices(MTD)	配置存储设备，需要该选项使 Linuxkeyiduqu 闪存卡（Flash、Card）存储器
Parallel port support	配置并口。如果不使用，可不选
Block devices	块设备支持
ATA/ATAPI/MFM/RLL support	IDE 硬盘和 ATAPI 光驱，纯 SCSI 系统且不使用这些接口，可以不选
SCSI device support	SCSI 仿真设备支持
Multi-devicesupport(RAID and LVM)	多设备支持（RAID 和 LAM）
Fusion MPT device support	MPT 设备支持
IEEE 1394(FireWire)support	IEEE 1394（火线）
I2O device support	I2O 设备支持。如果有 I2O 界面，必须选中。是由于智能 I/O 系统的标准接口
Network device support	内核在没有网络支持选项的情况下甚至无法编译。是必选项。
ISDN subsystem	综合数字业务网
Input device support	输入设备，包括鼠标、键盘等
Character devices	字符设备，包括虚拟终端、串口、并口等设备
I2C support	用于监控电压、风扇转速、温度等。
Hardware Monitoring support	需要 I2C 的支持
Misc devices	杂项设备
Multimedia Capabilities Port drivers	多媒体功能接口驱动
Multimedia devices	多媒体设备
Graphics support	图形设备/显卡支持
Sound	声卡
USB support	USB 接口支持配置
MMC/SD Card support	MMC/SD 卡支持

**步骤 3:** 配置选择内核定制，选择自己需要的功能。按键盘空格键进行选择\*表示选定直接编译进内核，M 表示选定模块编译为动态加载模块。在这里，以让内核支持 ntfs 文件系统为例。



1. 在 make menuconfig 命令打开的窗口中选择 Files systems。如下图：

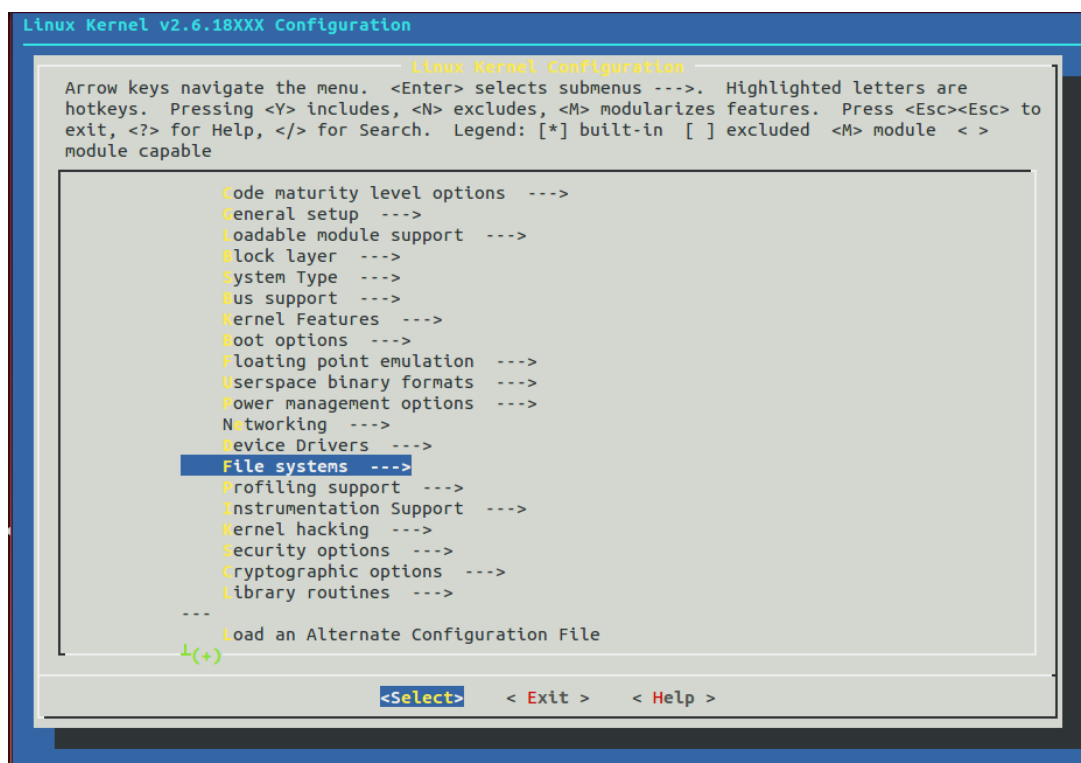


图 6-4 文件系统配置

2. 敲回车后，继续选择能支持 ntfs 选择系统类型的选项。如下图：

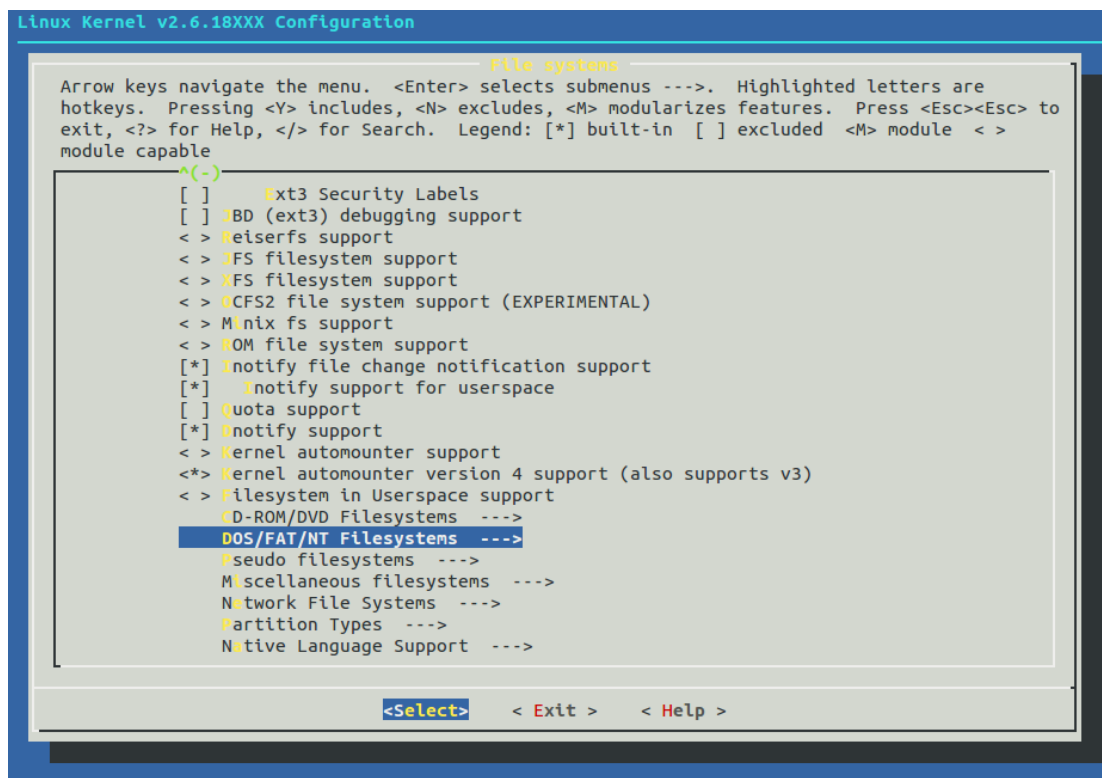


图 6-5 文件系统选项

3. 继续回车键，最后选择我们需要的 ntfs 文件系统类型。如下图：

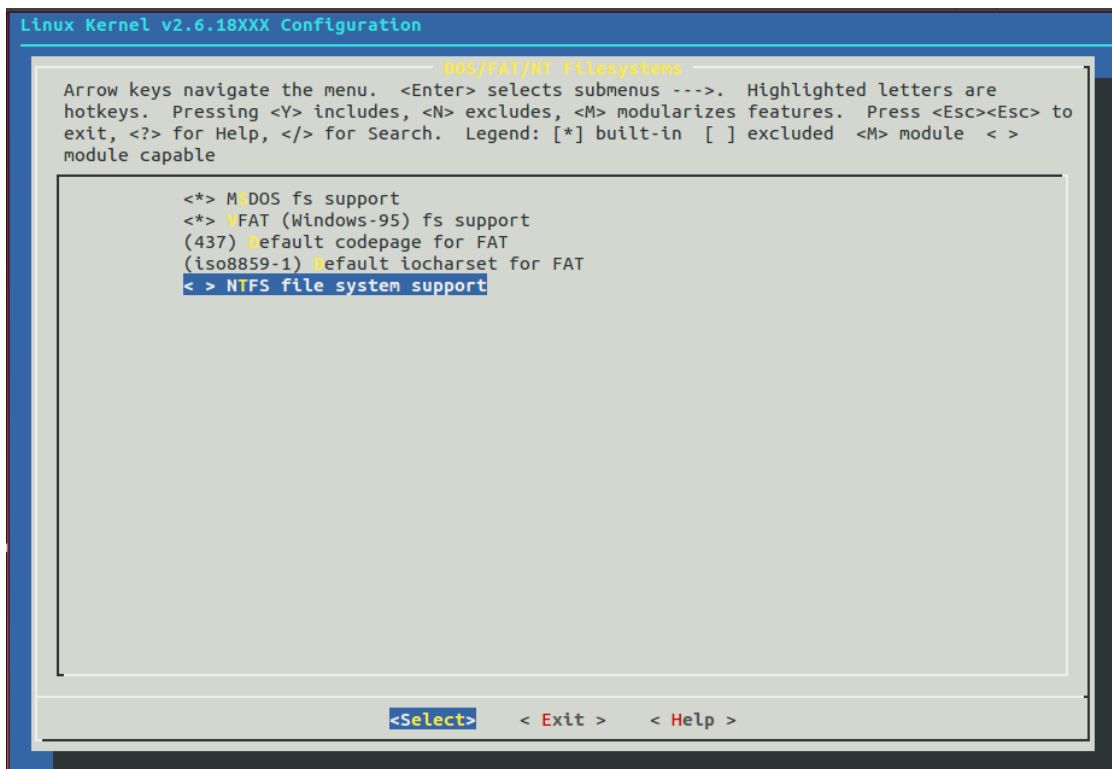


图 6-6 NTFS 文件系统选项

4. 按空格键选择编译进内核。并在键盘上左右键移动光标退出键按钮，按回车键不断退出。如下图所示：

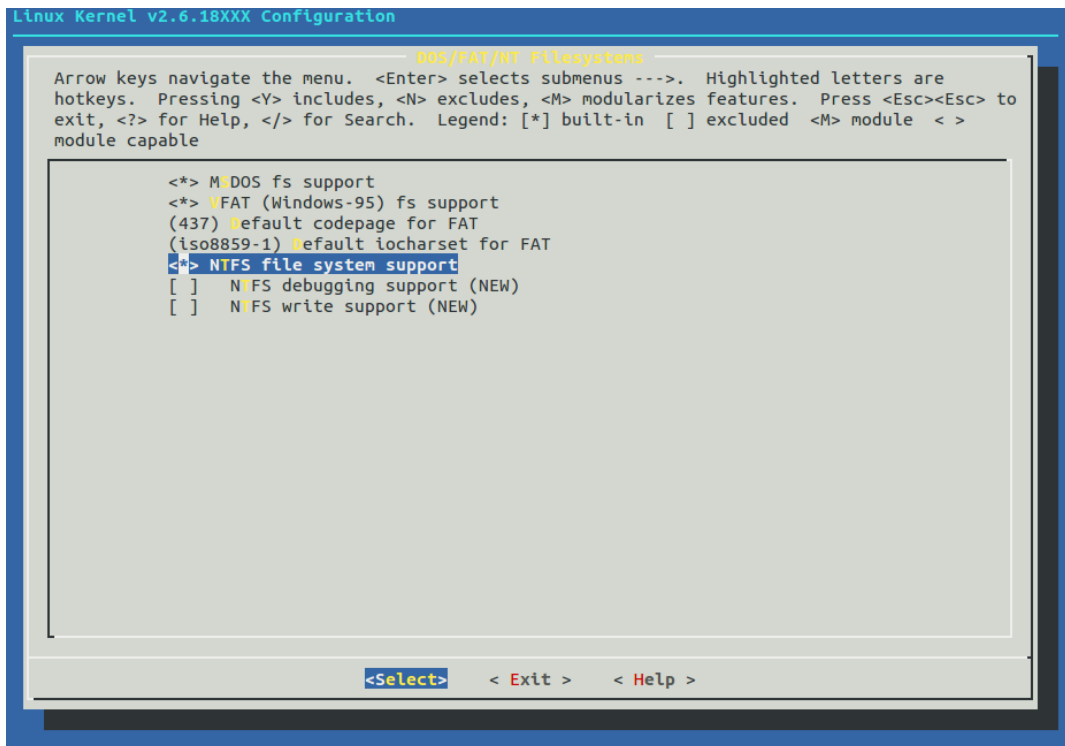


图 6-7 配置为编译进内核

5. 不断回车后，出现是否保存界面，选择 yes 保存配置，回车键退出。

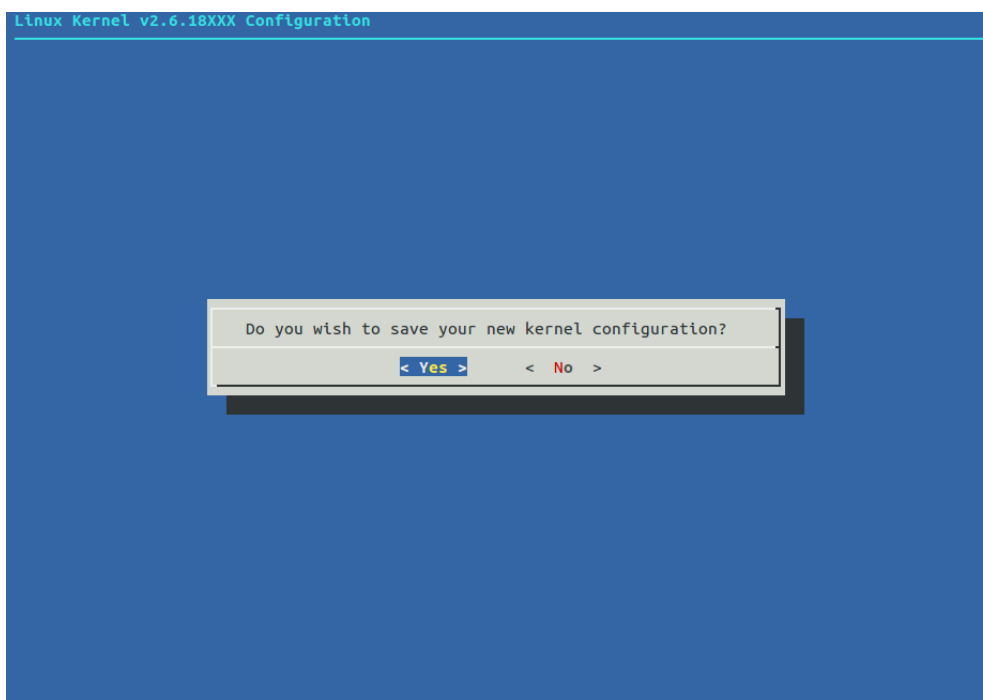


图 6-8 配置退出保存界面

#### 步骤 4：编译内核镜像

编译内核时候一般需要 root 权限，对特定用户加 sudo 命令即可。在内核目录下使用命令 `make uImage`。回车键后内核开始编译，等待出现 `Image arch/arm/boot/uImage is ready` 表示编译结束。编译好后再目录 `arch/arm/boot/` 下生成一个 uImage 二进制文件。如图所示：

```
root@ubuntu:/kernel/kernel-for-mceb# make uImage
HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf -s arch/arm/Kconfig
arch/arm/mach-davinci/Kconfig:44:warning: defaults for choice values not supported
drivers/video/Kconfig:1552:warning: 'select' used by config symbol 'FB_FSL_DIU' refer to undefined symbol 'PPC_LIB_RHEAP'
CHK     include/linux/version.h
SYMLINK include/asm-arm/arch -> include/asm-arm/arch-davinci
Generating include/asm-arm/mach-types.h
CHK     include/linux/utsrelease.h
UPD     include/linux/utsrelease.h
CC      arch/arm/kernel/asm-offsets.s
GEN     include/asm-arm/asm-offsets.h
HOSTCC  scripts/genksyms/genksyms.o
HOSTCC  scripts/genksyms/lex.o
scripts/genksyms/lex.c:1228:12: warning: 'input' defined but not used [-Wunused-function]
static int input()
        ^
HOSTCC  scripts/genksyms/parse.o
HOSTLD  scripts/genksyms/genksyms
CC      scripts/mod/empty.o
HOSTCC  scripts/mod/mk_elfconfig
MKELF   scripts/mod/elfconfig.h
```

图 6-9 开始内核编译

```
LD      arch/arm/boot/compressed/vmlinux
OBJCOPY arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
UIIMAGE arch/arm/boot/uImage
"mkimage" command not found - U-Boot images will not be built
Image arch/arm/boot/uImage is ready
root@ubuntu:/kernel/kernel-for-mceb#
```

图 6-9 编译生成的内核镜像

这就是利用编译进内核的方法编译生成的内核镜像，可以移植到实验板子上。对于一般的要求，利用编译进内核的方法就足够了。不过对于许多实验和工程的要求，为了减轻内核的负担，往往是需要利用动态加载的方法。下面继续实验来熟悉动态模块编译的方法。

#### 步骤 5：将 NTFS 文件系统配置成模块方式

按步骤 3 中的方法，将 NTFS file system support 嵌满\*改成为 M，即将 ntfs 文件系统配置成为模块加载形式。如下图所示：

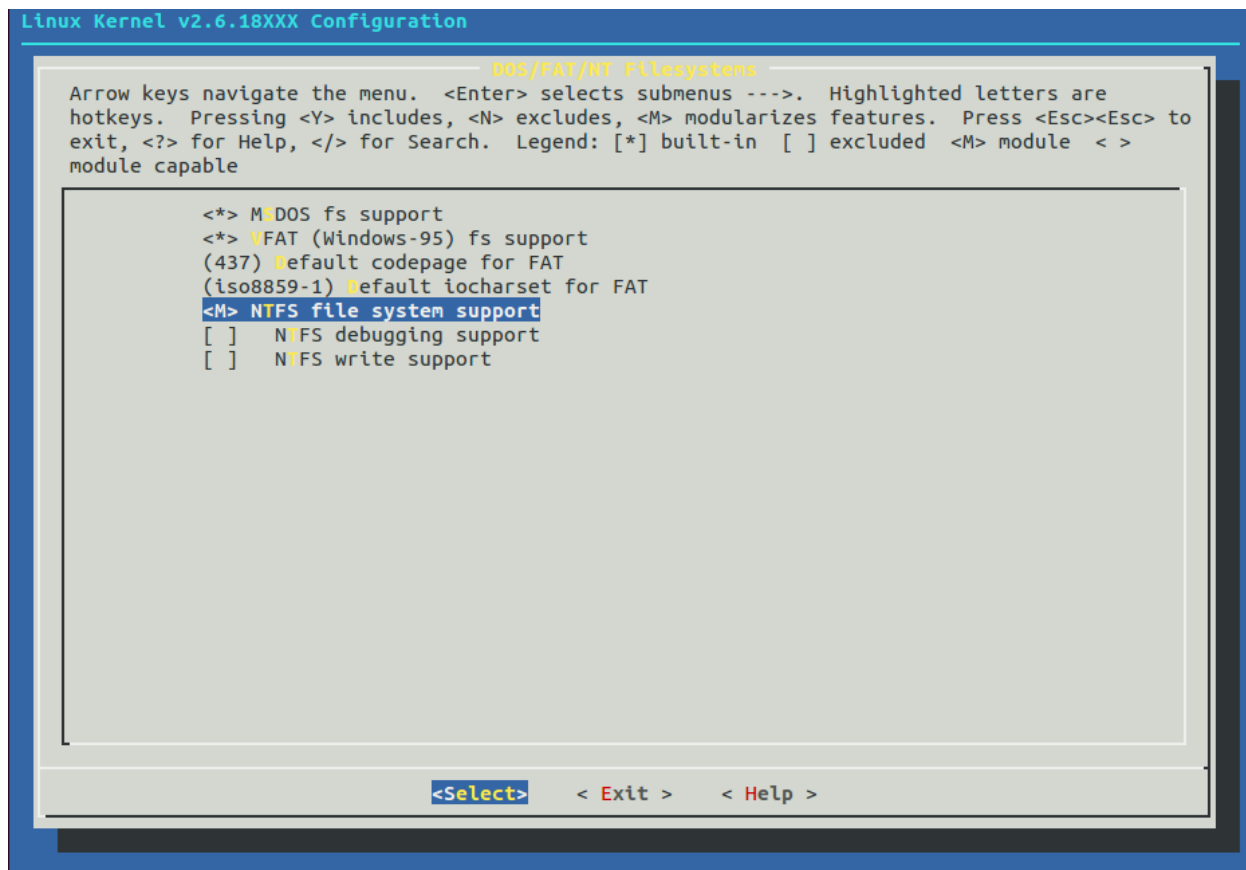


图 6-10 内核配置为模块方式

**步骤 6：**重新编译内核镜像由于编译生成的模块需要被加载到内核中才能使用，而开始编译生成的内核是一个可以支持 ntfs 文件系统的内核。可以先执行 make

clean, 清除刚才生成的镜像, 然后执行 `make uImage` 生成一个新内核镜像。即新生成的内核不支持 ntfs 文件系统, 从而可以将 ntfs 文件模块添加进去, 使内核可以支持 ntfs 文件系统。

```
root@ubuntu:/kernel/kernel-for-mceb# make clean
CLEAN arch/arm/boot/compressed
CLEAN arch/arm/boot
CLEAN /kernel/kernel-for-mceb
CLEAN arch/arm/kernel
CLEAN drivers/char
CLEAN drivers/video/logo
CLEAN init
CLEAN kernel
CLEAN lib
CLEAN usr
CLEAN include/asm-arm/mach-types.h include/asm-arm/arch include/asm-arm/.arch vmlinux System.map .tmp_kallsyms1.o .tmp_kallsyms1.S .tmp_kallsyms2.o .tmp_kallsyms2.S .tmp_vmlinux1 .tmp_vmlinux2 .tmp_System.map
rm: cannot remove 'include/asm-arm/arch': Is a directory
make: *** [clean] Error 1
root@ubuntu:/kernel/kernel-for-mceb#
```

图 6-11 清除刚才生成的镜像

```
LOGO drivers/video/logo/clut_vga16.c
LOGO drivers/video/logo/logo_linux_vga16.c
LOGO drivers/video/logo/logo_linux_clut224.c
LOGO drivers/video/logo/logo_punuo_clut224.c
LD drivers/video/built-in.o
LD drivers/built-in.o
HOSTCC lib/gen_crc32table
GEN lib/crc32table.h
CC lib/crc32.o
LD lib/built-in.o
GEN .version
CHK include/linux/compile.h
LD .tmp_vmlinux1
KSYM .tmp_kallsyms1.S
AS .tmp_kallsyms1.o
LD .tmp_vmlinux2
KSYM .tmp_kallsyms2.S
AS .tmp_kallsyms2.o
LD vmlinux
SYSMAP System.map
SYSMAP .tmp_System.map
OBJCOPY arch/arm/boot/Image
Kernel: arch/arm/boot/Image is ready
AS arch/arm/boot/compressed/head.o
GZIP arch/arm/boot/compressed/piggy.gz
AS arch/arm/boot/compressed/piggy.o
CC arch/arm/boot/compressed/misc.o
In file included from arch/arm/boot/compressed/misc.c:30:
include/asm/arch/uncompress.h:27: warning: conflicting types for built-in function 'putc'
LD arch/arm/boot/compressed/vmlinux
OBJCOPY arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
UIMAGE arch/arm/boot/uImage
"mkimage" command not found - U-Boot images will not be built
Image arch/arm/boot/uImage is ready
root@ubuntu:/kernel/kernel-for-mceb#
```

图 6-12 模块编译结束

## 四、心得与体会

通过这次内核编译实验, 对 ARM 的内核结构有了更为深入的理解, 并能够学会正确配置内核, 编译内核镜像, 对于其中重要的参数也有更为全面的认识和理解。