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. 实验设备

1．硬件：PC 机；dm365 系统教学试验箱；串口线；网线。

2．软件：PC 机操作系统；Windows 下超级终端 putty。

3．环境：ubuntu 系统版本 12.04；内核版本 kernel-for-mceb。

四. 预备知识

1. 概述

1.1 什么是 Linux 内核

内核是操作系统的核心部分，为应用程序提供安全访问硬件资源的功能。直接操作计算机硬件是很复杂的，内核通过硬件抽象的方法屏蔽了硬件的复杂性和多样性。通过硬件抽象的方法，内核向应用程序提供统一和简洁的接口，应用程序设计复杂度降低。实际上，内核可以看成是一个资源管理器，内核管理计算机中所有的硬件资源和软件资源。

1.2 Linux 内核版本

LInux 内核版本采用两个分割的“.”点，形式如”X.Y.Z”来表示。其中 X 表

示主版本号，Y 表示次版本号，Z 表示补丁号。奇数代表不稳定版本，偶数代表稳定版本。Linux 内核的官方网站为 http:www.kernel.org。该站点提供各种版本的代码和补丁，用户可以根据需要自由下载。试验箱所用的内核是基于 2.6.18 版本改进来的，并命名为 kernel-for-mecb。考虑到后续实验如内核的移植等需要，实验采用 kernel-for-mceb 内核。

2. 基本原理

2.1 内核配置和编译

内核编译主要分成配置和编译两部分。其中配置是关键，许多问题都是出在配置上。Linux 内核编译配置提供多种方法。如：

#make menuconfig　//基于图形工具界面

#make config　//基于文本命令行工具，不推荐使用

#make xconfig　//基于 X11 图形工具界面

由于对 Linux 还处在初学阶段，所以选择了简单的配置内核方法，即 make menuconfig。在终端输入 make menuconfig，等待几秒后，终端变成图形化的内核配置界面。进行配置时，大部分选项使用其缺省值，只有一小部分需要根据不同的需要选择硬件介绍。同时内核还提供动态加载的方式，为动态修改内核提供了灵活性。

2.2 内核编译系统

Linux 内核的复杂性，使其需要一个强大的工程管理工具。在 Linux 中，提供了 Makefile 机制。Makefile 是整个工程的编译规则。一个工程中源文件不计其数，按其类型、功能、模块被放在不同的目录中。Makefile 定义了一系列的规则来指定哪些文件需要先编译，那些文件需要后编译，哪些文件需要重新编译甚至进行更复杂的操作。Makefile 带来的直接好处就是自动化编译，一当写好，只要一个 make 命令，整个工程自动编译，极大提高效率。

内核编译时候通过 Makefile 规则将不同的文件进行整合。系统中主要有五种不同类型的文件，其类型和作用如下表所示：

表 1 编译的文件类型与作用表

文件类型	作用
Makefile	顶层 Makefile 文件

.config	内核配置文件
arch/$(ARCH)/Makefile	机器体系 Makefile 文件
scripts/Makefile.*	所有内核 Makefile 共用规则
Kbuild Makefile	其他 makefile 文件

表中.config 即是内核配置的文本文件。它记录了了文件的配置选项，可直接对其进行修改，只是较为繁琐，故不推荐使用。事实上，使用其他方式配置的文件最终都会保存到.config 中，换言之，内核配置就是围绕着.congfig 文件进行的。

内核编译的时候，顶层的 Makefile 文件在开始编译子目录下的代码之前，设置编译环境和需要用到的变量。顶层 Makefile 文件包含着通用部分，arch/$(ARCH)/Makefile 包含架构体系所需设置，其中也会设置一些变量和少量的目标。实验文档结尾有部分编译源码供参考。

3. 硬件介绍

理论上来说，内核作为操作系统核心只是软件。但由于内核统一管理着硬件，在这里就简要介绍下试验箱的主体硬件环境。DM365 核心板以其核心处理器芯片 TMS320DM365 为基础，高度集成了众多组件，如 H.264、MPEG-4、MPEG-2、MJPEG 与编解码器等。而 TMS320DM365 内部集成了：ARM 核子系统(ARMSS)；ARM926 RISC CPU 核与相关的存储器；Video 视频处理子系统(VPSS)：包括 Video Processing Front End (VPFE)，Image Input 和 Image Processing Subsystem 图像处理系统，和 Video Processing Back End (VPBE) Display Subsystem 等。并且有 DD2、NAND FLASH 、复位芯片、以太网控制芯片、I2C 总线通信、USB 控制器等外围设备以及丰富的硬件接口。内核统一管理着这些硬件设备。试验箱主要硬件框图如下图所示：

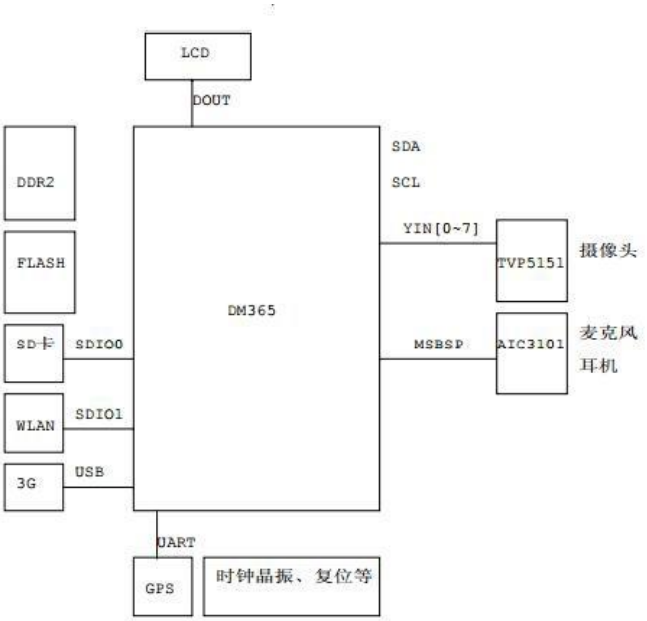

图 1　试验板子框图

4. 软件介绍

4.1 内核子目录

Linux 内核在不断升级中，以实验的所用基于 2.6.18 的内核来说，代码数量已经达到几百万行。如此庞大的代码量，使得内核源码需要组织有序，以便于学习和维护。Linux 内核把不同功能分成不同的子系统的方法，通过一种整体的结构把各种功能集合在一起，来解决了这个问题，同时提高工作效率。在服务上可以直接查看内核子目录。如下图所示：

```
ny@ubuntu:~/kernel-for-mceb$ ls
@          crypto          fs       lib              mm             scripts          vmlinux
1          cscope.in.out   include  localversion     Module.symvers security
arch       cscope.out      init     localversion-mvl mvl_patches    sound
block      cscope.po.out   ipc      ltt              net            System.map
COPYING    Documentation   Kbuild   MAINTAINERS      README         uImage_make_mceb
CREDITS    drivers         kernel   Makefile         REPORTING-BUGS usr
ny@ubuntu:~/kernel-for-mceb$
```

图 2　内核子目录

图中，内核目录下蓝色文件夹中代表着不同子目录，包含着不同功能的实现代码。主要子目录介绍如下：

表 2　内核子目录功能表

子目录	功能
/include	头文件
/init	内核的初始化代码

/arch	所有硬件结构特定的内核代码
/drivers	设备驱动程序代码
/fs	各种支持的文件系统
/net	内核的网络连接代码
/mm	子目录包含了所有内存管理代码
/kernel	主内核代码
/block	部分块设备驱动
/crypto	常用加密和散列算法代码
/Documentation	内核各部分的通用解释和注释
/ipc	进程间通信的代码
/scripts	配置内核文件的脚本和应用程序代码
/sound	常用音频设备驱动程序
/security	安全框架代码
/usr	用于支持用户空间代码

4.2 内核接口

早期计算机由于资源的局限性，应用程序可以直接在硬件上运行。但是随着计算机性能不断提高，硬件和软件都变得复杂的情况下，需要一个统一的管理程序，这就是操作系统，而内核就是操作系统最核心的部分。在操作系统中，内核提供丰富的接口，尤其是为应用程序提供的接口，可以使得应用程序达到访问硬件的目的。

图 3 内核接口

从图中可以看出，内核下面直接管理者硬件，向上提供系统调用接口。其中，GNU C 库是一种按照 LGPL 许可协议发布的 C 的编译程序，反映着内核的开源、免费的特性。内核接口中提供丰富的编程接口函数。

4.3 内核接口函数

内核中接口函数纷繁复杂，可以有选择地进行了解。在这里，以最常用的驱动字符设备为例，重点介绍一些常用的接口函数。例表如下：

表 3　字符设备常见接口函数

alloc_chrdev_region()	自动分配一个主设备号及基于此主设备号的若干个连续的指定数量的次设备号
register_chrdev_region()	注册连续的若干个设备号
unregister_chrdev_region()	注销注册的设备号
cdev_alloc()	分配一个表示字符设备的 cdev 结构体
cdev_init()	初始化由字符设备的 cdev 结构体
cdev_add()	添加字符设备到系统中
cdev_del()	从系统中删除字符设备

5. 内核编译工具

Gcc 是为 GNU 操作系统上专门编写的一款编译器，编译内核需要此编译环境。服务器上一般会事先安装好，可以事先检测一下，检测命令为 gcc -v。如下图所示：

```
hy@ubuntu:~$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/i686-linux-gnu/4.6/lto-wrapper
Target: i686-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu/Linaro 4.6.3-1ubuntu5'
 --with-bugurl=file:///usr/share/doc/gcc-4.6/README.Bugs --enable-languages=c,c++,for
tran,objc,obj-c++ --prefix=/usr --program-suffix=-4.6 --enable-shared --enable-linker
-build-id --with-system-zlib --libexecdir=/usr/lib --without-included-gettext --enabl
e-threads=posix --with-gxx-include-dir=/usr/include/c++/4.6 --libdir=/usr/lib --enabl
e-nls --with-sysroot=/ --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdc
xx-time=yes --enable-gnu-unique-object --enable-plugin --enable-objc-gc --enable-targ
ets=all --disable-werror --with-arch-32=i686 --with-tune=generic --enable-checking=re
lease --build=i686-linux-gnu --host=i686-linux-gnu --target=i686-linux-gnu
Thread model: posix
gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5)
```

图 4　检测 gcc 编译工具

如上图所示，输入命令后，会显示版本号，这里用到的是 4.6.3。Gcc 编译器 能将 C、C++语言源程序、汇程式化序和目标程序编译、连接成可执行文件。

上面所说的将代码变成可执行文件的过程就叫做编译。Linux 系统中，提供 的是 GNU make 工具。GNU make 是一个工程管理器，专门负责管理、维护较多 文件的处理，实现自动化编译。如果一个工程项目中，有成百上千个代码源文件， 若其中一个或多个文件进过修改，make 就需要能够自动识别更新了的代码。故 其编译需要一定的规则，都定义在文件 makefile 中。

五. 实验步骤

步骤 1： 登录服务器以用户名登录服务器，进入学生的目录 stX，找到内核目录，移动物联网试验箱的内核基于 2.6.18 改进来的 kernel-for-mceb。

```
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 ./
```

解压缩 kernel-for-mceb.tar 软件包，解压缩后会出现 kernel-for-mceb 文件夹：

```
#tar   zxvf   kernel-for-mceb.tar.gzcd
```

步骤 2： 进入内核进行配置

在配置之前首先输入命令 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，那么请先运行：make mrproper 清除以前的配置，回到默认配置。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：

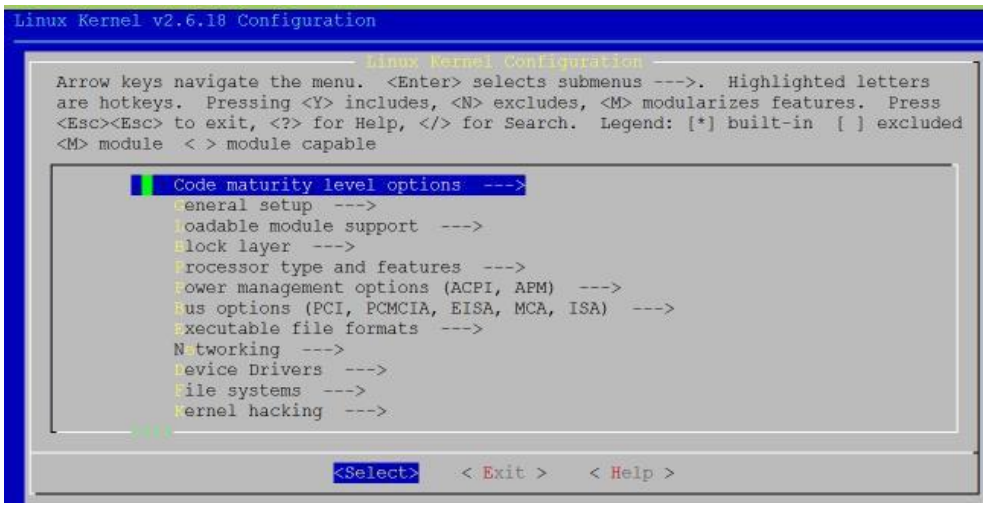

图 6　配置内核

编译内核时候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。

表 4　驱动选项表

选项	说明
MemoryTechnologyDevices(MTD)	配置存储设备，需要该选项使 Linux 可以读取闪存卡等（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到 Files systems。如下图：

图 7 文件系统配置

2．敲回车后，继续选择能支持 ntfs 文件系统类型的选项。如下图：

图 8 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：

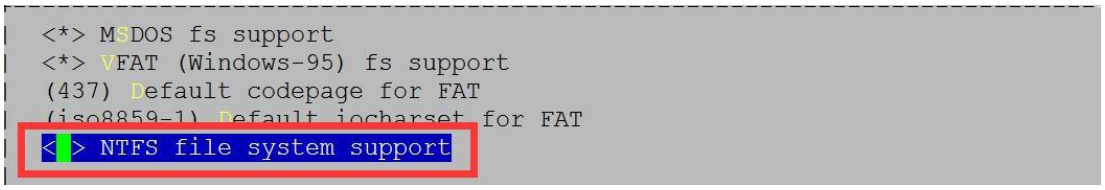

图 9　ntfs 文件系统选项

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

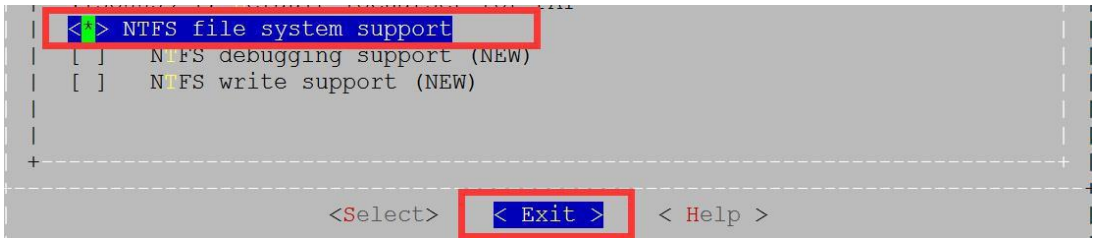

图 10 配置为编译进内核

5．不断回车键后，出现是否保存界面，选择 yes 保存配置，回车键退出。

图 11　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，对于特定用户加 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，等到出现 Image arch/arm/boot/uImage is ready 表示编译结束。编译好后在目录 arch/arm/boot/下生成一个 uImage 二进制文件。如图所示。

图 12　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

按照步骤 3 中的方法，将 NTFS file system support 前面*改变成为 M，即将 ntfs 文件系统配置成为模块加载形式。如下图所示：

图 13 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。 这就是模块编译的方法。关于查看、加载以及卸载模块的方法可以参考其他实验。

以上介绍的就是内核编译方法。以添加支持 ntfs 文件系统为例，介绍了两种方法：一种是直接编译进内核中，另一种是编译成模块加载到内核中。

附录：部分 makefile 源码解释

```
#版本基本信息
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 18
EXTRAVERSION =-plc
NAME=Avast! A bilge rat!
#####################################################################
MAKEFLAGS += --no-print-directory      #不要再屏幕上打印"Entering     directory..",始终被自动的传递给所有的子 make。
#####################################################################
ifdef V     #V=1。
      ifeq ("$(origin V)", "command line")    #函数 origin 指示变量是哪里来的。
        KBUILD_VERBOSE = $(V)     #把 V 的值作为 KBUILD_VERBOSE 的值。
       Endif
endif
  ifndef KBUILD_VERBOSE    #即默认我们是不回显的，回显即在命令执行前显示要执行的命令。
       KBUILD_VERBOSE = 0
endif
#####################################################################
ifdef    SUBDIRS
  KBUILD_EXTMOD ?= $(SUBDIRS)       #条件操作命令。
endif
Ifdef    M   #M 用来指定外在模块目录。
    ifeq ("$(origin M)", "command line")
        KBUILD_EXTMOD := $(M)
     endif
 Endif
#####################################################################
ifeq ($(KBUILD_SRC),)    #变量，是否进入下一层。
#####################################################################
```

```
ifdef O   #把输出文件放在不同的文件夹内。
   ifeq ("$(origin O)", "command line")
      KBUILD_OUTPUT := $(O)     #用于指定我们的输出文件的输出目录。
   endif
Endif
#################################################################
PHONY := _all   #默认目标是全部。
_all:
ifneq ($(KBUILD_OUTPUT),)    #检测输出目录。
#################################################################
saved-output := $(KBUILD_OUTPUT)
KBUILD_OUTPUT := $(shell cd $(KBUILD_OUTPUT) && /bin/pwd)    #测试目录是否存在，存在则赋给 K BUILD_OUTPUT。
$(if $(KBUILD_OUTPUT),, \     #这里的为空即表示输出目录不存在
      $(error output directory "$(saved-output)" does not exist)) )     #使用了 error 函数
PHONY += $(MAKECMDGOALS)    #将任何在命令行中给出的目标放入变量。
$(filter-out _all,$(MAKECMDGOALS)) _all: )    #表示要生成的目标。
      $(if $(KBUILD_VERBOSE:1=),@)$(MAKE) -C $(KBUILD_OUTPUT) \
      KBUILD_SRC=$(CURDIR) \
      KBUILD_EXTMOD="$(KBUILD_EXTMOD)" -f $(CURDIR)/Makefile $@
      #$@表示取消回显的意思。
#################################################################
skip-makefile := 1 #跳转目录所用
endif    #KBUILD_OUTPUT 结束处。
endif    #KBUILD_SRC 结束处
#################################################################
#以下设置编译连接的默认程序,都是变量赋值操作。
AS          = $(CROSS_COMPILE)as
LD          = $(CROSS_COMPILE)ld
```

```
CC          = $(CROSS_COMPILE)gcc
CPP         = $(CC) -E
AR          = $(CROSS_COMPILE)ar
NM          = $(CROSS_COMPILE)nm
STRIP       = $(CROSS_COMPILE)strip
OBJCOPY     = $(CROSS_COMPILE)objcopy
OBJDUMP     = $(CROSS_COMPILE)objdump
AWK         = awk
GENKSYMS    = scripts/genksyms/genksyms
DEPMOD      = depmod
KALLSYMS    = scripts/kallsyms
PERL        = perl
CHECK       = sparse
CHECKFLAGS      := -D  linux  -Dlinux -D  STDC  -Dunix -D  unix  -Wbitwise $(CF)
MODFLAGS    = -DMODULE
CFLAGS_MODULE   = $(MODFLAGS)
AFLAGS_MODULE   = $(MODFLAGS)
LDFLAGS_MODULE  = -r
CFLAGS_KERNEL   =
AFLAGS_KERNEL   =
###########################################################################
clean := -f $(if $(KBUILD_SRC),$(srctree)/)scripts/Makefile.clean obj   #清除设置。
```