友情提醒：

1. 本提纲仅适用于南京邮电大学李超老师的嵌入式系统开发课程。
2. 老师本人很认真地讲课，对于 C 语言和底层的讲述非常细致，如果可以请课堂仔细听讲，避免错过良师。
3. 实验课请动手，至少点亮 LED 灯！有问题请问老师！
4. 老师不会随便为难同学，请认真复习。根据提纲，作者期末获得 97 分。

考试携带：

1. 教材
2. 带有答案的实验报告
3. 测试题和答案

实验部分：

离不开 LED 和看门狗的实验，仔细理解做起来很容易。考试题仅会是这两种程序的变种。而这两种程序老师课堂至少说 5 遍，哪怕认真听一遍也不可能做不出！

最终声明：

若有题型改变和书本变革，本提纲不负责任！

嵌入式期末——理论部分

CH1：

1. 嵌入式系统的定义、特点：

定义：

书 P_1：以系统为中心，以计算机技术为基础，软硬件可剪裁，适应应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。

特点：

①微内核
②专用性强
③可裁剪
④专用操作系统的支撑
⑤专用的开发工具和环境

2. 嵌入式系统微处理器种类、典型代表：

书 P_2：

ARM 微处理器，MIPS，68K/Cold Fire、Power PC

举例列出一款 ARM7TDMI 微内核的嵌入式微处理器 **S3C44B0X**，ARM920T 微内核的嵌入式微处理器 **S3C2410**，ARM11 内核的嵌入式微处理器 **S3C6410**，并列举 2 款 64 位 ARM 微内核 **Cortex-A53**、**Cortex-A57**。

3. 嵌入式 OS 的种类及典型代表：

书 P_7：

VxWorks、RT-Linux、uClinux、ARM-Linux、Windows Phone、iOS、Android

4. 开发模式

书 P_{10}：

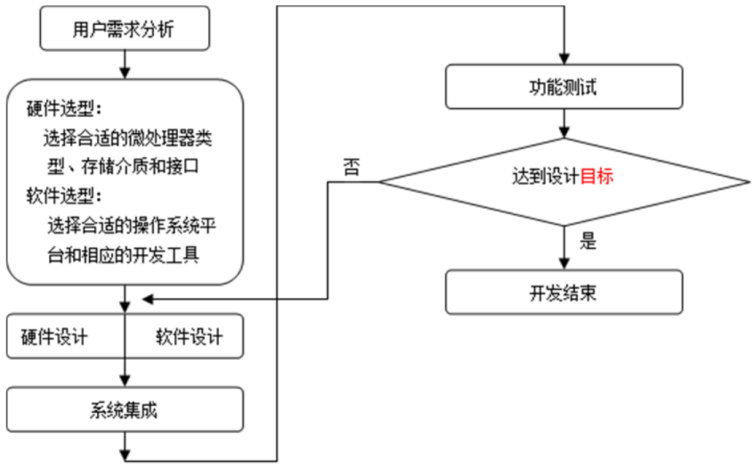
宿主机——目标板模式

了解基于 ARM 核的研究和商业运作模式。

§ 企业运行的模式— chipless 的生产模式
§ 公司既不生产芯片，也不设计芯片，而是设计出高效的 IP 内核
§ 全球有 103 家巨型 IT 公司在采用 ARM 技术，20 家最大的半导体厂商中有 19 家是 ARM 的用户

5. 开发流程

书 P_{10} 流程图：



CH2：

1. ARM 处理器的特点、构成

特点：

书 P14：

①体积小、功耗低、成本低、高性能。

②支持 Thumb（ 16 位） /ARM（ 32/64 位）双指令集，能够兼容 8 位和 16 位器件。

③具有大量的寄存器，因而指令执行速度快。

④绝大多数操作都在寄存器中进行， 通过 Load/Store 的体系架构在内存和寄存器之间传递数据。

⑤寻址方式简单。

⑥采用固定长度的指令格式。

ARM 指令格式：

（1）ARM 汇编程序由**机器**指令、**汇编**指令和**伪**指令构成。

（2）ARM 伪指令可以分为以下几类：**符号定义伪指令，数据定义伪指令，汇编控制伪指令，信息报告伪指令，宏指令以及其他伪指令**。

2. 存储体系：大/小端模式

书 P20：

大端模式：

在该种模式下，字数据的高字节存放在低地址中，而字数据的低字节部分则存放在高地址中。如图 2-5 所示。

小端模式：

小端模式与大端模式存储格式恰好相反，在小端模式下，低地址中存放字数据的低字节部分，而高地址存放的是字数据的高地址部分。如图 2-6 所示。

测试题：假设存储数据 0x12345678 于 ARM 微处理器内存 0X30000000 开始的位置，则 0X30000001 内存位置的数据为 __C__（A.0X12 B.0X34 C.0X56 D.0X78）（采用小端模式进行存储）

3. 异常处理模式

书 P17：

异常是 ARM 微处理器中非常重要的一个事件，它由内部或外部源产生而引起处理器处理。ARM 微处理器支持表 2-1 所列几种类型的异常。

表 2-1　ARM 体系中异常类型

异常类型	含义
复位	当处理器的复位引脚有效时（可以有软硬件引起），系统产生复位异常中断，程序跳转到复位程序处执行。
未定义指令	当 ARM 处理器或者协处理器执行到未定义指令时，产生未定义的指令异常中断，转到相应的处理程序处执行。
软件中断	软件中断由用户执行指令 SWI 而引起，可用于用户模式下的程序调用调用特权操作指令。
指令预取中止	若处理器预取的指令的地址不存在，或该地址不允许当前指令访问时，当该被预取的指令执行时，才会产生指令预取中止异常。
数据访问中止	如果数据访问指令的目标地址不存在，或者该地址不允许当前指令访问，处理器产生数据访问中止异常中断。
外部中断请求（IRQ）	当处理器的外部中断请求引脚有效，而且 CPSR 寄存器的 I 位为 0 时，处理器产生外部中断请求 IRQ 异常中断。系统的外设通常通过该异常
	中断请求中断服务。
快速中断请求（FIQ）	当处理器的快速中断请求引脚有效，并且 CPSR 寄存器的 F 位为 0 时，处理器产生外部中断请求 FIQ 异常中断。

练习：

一般情况下，ARM 微处理器异常处理模式共有___7___种，机器启动后第一条指令执行的是__A___（A.复位异常处理函数指令 B.中断异常处理指令 C.IRQ 异常处理指令 D.指令预取终止异常）。

4. 传参原则

书 P_{30}：

根据参数个数是否固定,可以将子程序分为参数个数固定的子程序和参数个数可变的子程序。这两种子程序的参数传递规则是不同的。

①参数个数可变的子程序参数传递规则

对于参数个数可变的子程序,当参数不超过 4 个时,可以使用寄存器 R0~R3 来进行参数传递,当参数超过 4 个时,还可以使用数据栈来传递参数。在参数传递时,将所有参数看做是存放在连续的内存单元中的字数据。然后依次将各名字数据传送到寄存器 R0,R1,R2,R3; 如果参数多于 4 个,将剩余的字数据传送到数据栈中,入栈的顺序与参数顺序相反,即最后一个字数据先入栈. 按照上面的规则,一个浮点数参数可以通过寄存器传递,也可以通过数据栈传递,也可能一半通过寄存器传递，另一半通过数据栈传递。

②参数个数固定的子程序参数传递规则

对于参数个数固定的子程序,参数传递与参数个数可变的子程序参数传递规则不同,如果系统包含浮点运算的硬件部件,浮点参数将按照下面的规则传递: 各个浮点参数按顺序处理;为每个浮点参数分配 FP 寄存器;分配的方法是,满足该浮点参数需要的且编号最小的一组连续的 FP 寄存器.第一个整数参数通过寄存器 R0~R3 来传递,其他参数通过数据栈传递。

调用函数 FUN(X,Y,Z)，则实参值分别通过 **r_0、r_1、r_2** 寄存器来进行传递，如果参数超过 4 个，则参数传递规则为**通过栈进行传递**。

5. R_{13}、R_{14}、R_{15}的作用

书 P_{16}：

R_{13}：用作堆栈指针

R_{14}：

作用①：用户模式下， R_{14} 用做链接寄存器（LR） ,存放子程序被调用时的返回地址。
作用②：异常处理模式下， R_{14} 用来保存异常的返回地址。
R_{15}：程序计数器，又被记做 PC。由于 ARM 采用了流水线机制，因此 PC 的值为当前指令地址的值加 8 个字节，也就是说， PC 指向当前指令的下两条指令的地址。

CH3：

1. S3C6410 微处理架构
2. 软件操作、硬件方式

非 OS：端口寄存器（读、写）
例子：
端口的地址为 **0x7F008800。**

```
#define rGPKCON0 (*((volatile int *)(0x7F008800)));
                 rGPKCON0 = 0x55aa
```

或

```
(*((volatile int *)(0x7F008800)))=0x55aa;
```

3. **LED 灯与看门狗（实验代码等见实验部分）**

看门狗的作用：书 P_{37}：
系统软件或者应用软件都可能出错，导致整个系统无法继续正常工作，此时必须让整个系统重启以恢复其原有的功能，在 S3C6410 微处理器中集成的看门狗定时器就可以实现这个功能。

CH4：

1. 交叉编译器核心部件

gcc、make、gdb 的安装。
在 PC 机上 Linux 系统编译使用的编译器名为 **gcc**，ARM 处理器嵌入式编译器名为 **arm-linux-gcc** 。

2. **gcc 的使用**

-o / -C / -l / -L / -I / -g /-O

书 P_{46} ：

◆ **-o**
功能：制定目标名称,缺省的时候,gcc 编译出来的文件是 a.out
例：

```
arm-linux-gcc -o hello.exe hello.c
```

◆ **-c**
功能：只对文件进行编译和汇编,但是并不进行连接，也就是说只把程序做成 obj 文件。
例:

```
arm-linux-gcc -c hello.c
```

将生成.o 的 obj 文件 。

◆ **-l，-I，-L**

-I：指定第一个寻找头文件的目录
-L：指定第一个寻找库文件的目录
-l：表示在库文件目录中寻找指定的动态库文件

◆ **-g**

编译器在编译的时候产生调试信息。

3. gdb 操作步骤 、常用命令。

书 P_{56} ：

常用命令：

命　令	描　述
file	装入想要调试的可执行文件.
kill	终止正在调试的程序.
list	列出产生执行文件的源代码的一部分.
next	执行一行源代码但不进入函数内部.
step	执行一行源代码而且进入函数内部.
run	执行当前被调试的程序
quit	终止 gdb
watch	使你能监视一个变量的值而不管它何时被改变.
break	在代码里设置断点，这将使程序执行到这里时被挂起.
make	使你能不退出 gdb 就可以重新产生可执行文件.
shell	使你能不离开 gdb 就执行 UNIX shell 命令.

（更多命令见 P_{57}）

运行 gdb 调试程序时通常使用如下的命令：

gdb 文件名

操作步骤：

（1）运行程序

在命令行上键入 gdb 并按回车键就可以运行 gdb 了，如果一切正常的话，gdb 将被启动并且将在屏幕上看到类似的内容：

GDB is free software and you are welcome to distribute copies of it under
certain conditions; type "show copying" to see the conditions.
There is absolutely no warranty for GDB; type "show warranty" for details.GDB
4.14 (i486-slakware-linux), Copyright 1995 Free Software Foundation, Inc.
(gdb)

当启动 gdb 后，可以在命令行上指定很多的选项。也可以以下面的方式来运行 gdb：

gdb <fname>

当用这种方式运行 gdb，程序员能直接指定想要调试的程序。 这将告诉 gdb 装入名为 fname 的可执行文件。

（2）设置/显示命令行参数

设置命令行形式：

```
set args <arg list>
```

显示命令行形式：

```
show args
```

（3）程序运行相关命令

……

gdb 远程调试 **书 P_{68}：**

需要远程调试程序时，首先要对你需要进行远程调试的程序做一些改造，在程序的开始插入 set_debug_traps()和 break_point()函数，然后重新编译，并且将 GDB Stub，你的程序，

还有串口驱动程序等能一起连接在一起成为新的可执行程序， 将它的一份拷贝到远程主机上。

然后按照如下的步骤：

ν 将两台机器用串口线连接起来

ν 将需要调试的程序拷贝到远程主机

ν 在本地主机启动 GDB，读入需要调试的程序的符号表和程序代码

ν 使用 target remote 命令建立和远程主机的连接

ν 然后就像和使用一般的 GDB 一样进行程序的调试了。

4. Code::Blocks 作用和基本操作

书 P_{68}：

作用：

Code::Blocks 是一个**开放源码的全功能的跨平台 C/C++集成开发环境**。 Code::Blocks 是开放源码软件。Code::Blocks 由纯粹的 C++语言开发完成。

基本操作：

见**书（P_{69}-P_{77}）**完整流程。

CH5：

1. 嵌入式 Linux 系统核心部件

嵌入式 Linux 操作系统包括 **bootloader 、 内核 、文件系统** 三部分组成。

2. bootloader 的功能、种类

bootloader 的功能： **①引导操作系统内核启动 ②提供辅助命令工具** 。

常见 bootloader：**redboot、lilo、uboot、grub、etherboot、vivi**。

3. U-boot 常用命令

书 P_{82} ：

复习题：在 uboot 中，**打印开发板上环境变量值**的命令为 **printenv** ，**设置 IP 地址**为 192.168.1.1 的命令为 **setenv ipaddr 192.168.1.1** ，假如嵌入式内核名为 vmlinux，通过 **tftp 加载内核**的命令为 **tftp vmlinux 内存地址** ，**启动嵌入式 Linux 内核**的命令为 **bootm** 。

其他命令：参考教材。

4. bootloader、内核、文件系统构建步骤

嵌入式 Linux 内核裁剪命令为 **make menuconfig**；裁剪完成后生成配置选择文件 **.config**；编译内核命令为 **make zImage** 。

构建嵌入式 Linux 系统时，烧写 bootloader 的工具为 **hjtag** ；烧写 Linux 内核一般使用 **A** （A.bootloader B.文件系统 C.交叉编译器）来烧写，烧写文件系统一般使用 **A** （A.bootloader B.文件系统 C.交叉编译器）来烧写。

从 C 语言角度来理解，嵌入式 Linux 内核仅是众多 **函数** 的集合体，其有一个类似于 main 的函数，名称为 **start_kernel** 。

启动 linux 后常用的命令如 ifconfig、cp、ls 等通常位于 C（A.bootloaderB.内核 C.文件系统）中；嵌入式 Linux 环境下生成文件系统的常用工具为 mkyaffs2img；以 yaffs 方法为例，将嵌入式文件夹压缩成一个 yaffs2 格式的文件系统命令为 mkyaffs2img 文件夹　镜像文件名　。

具体内容见实验部分。

CH6：

1. 开发板和 PC 机传输文件的方式

将编译好的可执行文件下载到目标板目前主要四种方式：
第一种：通过 ftp 传送文件到开发板(**推荐使用**)
第二种：复制到介质(如优盘)
第三种：通过串口传送文件到开发板
第四种：通过 NFS(网络文件系统)直接运行

2. 移植步骤：configure、make、make install

嵌入式 Linux 应用程序移植常用 configure 命令生成 Makefile,一般来说，指定安装目录为/opt/soft 的命令为 ***./configure　--prefix=/opt/soft*** ；如果待生成的可执行目标板为 ARM 处理器，则命令为 ***./configure　--prefix=/opt/soft　--HOST=arm-linux*** 。

其余内容见实验部分或 **书 P102**

3. Sqlite 命令使用方式

书 P105：

sqlite 操作的基本步骤包括：

- 打开数据库
- 执行相关操作，如插入记录、删除记录、查询等
- 关闭数据库

（1）打开数据库

函数原型：

```
int sqlite3_open(const char *filename, sqlite3 **ppDb);
```

filename：待打开(创建)的数据库文件名；

ppDb： sqlite3 数据库句柄的指针

（2）操作数据库

sqlite 可以提供了接口，可以把 SQL 操作语句直接嵌入到 sqlite 函数中来执行。

函数原型：

```
int sqlite3_exec( sqlite3* ppDb, const char *sql, int (*callback)(void*,int,char**,char**) , void*,char **errmsg);
```

第 1 个参数是 open 函数得到的指针。第 2 个参数 constchar*sql 是一条 sql 语句，以\0 结尾。 第 3 个参数 sqlite3_callback 是回调，当这条语句执行之后， sqlite3 会去调用这个函数。第 4 个参数 void*是你所提供的指针，你可以传递任何一个指针参数到这里，这个参数最终会传到回调函数里面，如果不需要传递指针给回调函数，可以填 NULL。等下我们再看回调函数的写法，以及这个参数的使用。 第 5 个参数 char** errmsg 是错误信息。

（3）关闭数据库

数据库不使用后需要关闭。

函数原型：

int sqlite3_close(sqlite3 *ppDb);

测试题：

嵌入式数据库经常使用 sqlite，为可以在开发板上运行的轻型数库。通常情况下 sqlite 提供 ***命令*** 和 ***函数编程*** 两种使用方式。

在 sqlite 下创建数据库 stu.db 命令为 ***sqlite3 stu.db*** ；创建包含学号，姓名和电话号码的数据表 StuPhone 命令为 ***create table StuPhone(id integer primary key, name text, phoneNo text);*** ；

插入 10010201,wangming,13900008888 记录的命令为 ***insert into StuPhone values(10010201,wangming,13900008888)*** 。

4. QT 的信号与插槽机制

书 P_{108}：

连接：

所有从 QObject 或其子类（例如 Qwidget）派生的类都能够包含信号和槽。因为信号与槽的连接是通过 QObject 的 connect() 成员函数来实现的。

connect(sender, SIGNAL(signal), receiver, SLOT(slot));

其中 sender 与 receiver 是指向对象的指针， SIGNAL() 与 SLOT() 是转换信号与槽的宏。

测试题：

在嵌入式 Linux 开发过程中，EmbeddedQT 通常情况下用于 **C**（A.内核代码 B.驱动代码 C.图形应用程序）设计。EmbeddedQT 采用工具开发包的形式提供给用户，一般情况下包括**图形设计器** 、**QT 的 C++类库**和 Makefile 制作工具，字体国际化工具等。

QT 开发中采用 **信号和插槽** 机制来连接两个对象之间的通讯，假设对象 a 的 clicked 信号和对象 b 的 handleFunction() 相关联，对应语句为：***connect(&a,SIGNAL(clicked()),&b,SLOT(handleFunction()));***

QT 的优点：

- 优良的跨平台特性
- 面向对象
- 丰富的 API

QT 实现 LED 灯亮灭：

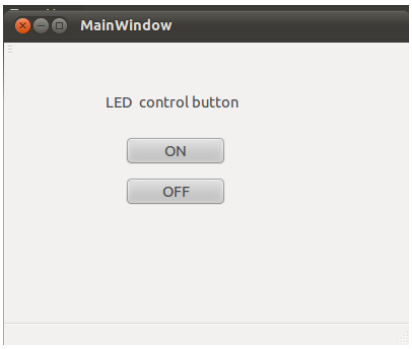
假设有嵌入式 LED 报警灯驱动设备文件/dev/led，点亮 led 灯调用函数 ioctl(fd,LEDON), 熄灭 led 灯调用函数 ioctl(fd,LEDOFF),请设计 QT 应用程序完成如下任务，点击 ON 按钮点亮 LED 灯，点击 OFF 按钮熄灭 LED 灯。写出这两个按钮的相关联的函数。假设 ON 按钮名称为 m_on,OFF 按钮名称为 m_off。

答案：

关联函数：

```
connect(m_Win,SIGNAL(clicked()),m_on,SLOT(LedOnFun()));
connect(m_Win, SIGNAL(clicked()),m_off,SLOT(LedOffFun()));
```

```
void LedOnFun( )
{
    int fd;
    fd=open("/dev/leddev",O_RDWR);
    ioctl(fd,LEDON);
    close(fd);
}
void LedOffFun()
{
    int fd;
    fd=open("/dev/leddev",O_RDWR);
    ioctl(fd,LEDOFF);
    close(fd);
}
```



详细阐述：书 P108：

CH7：

1. 设备驱动的概念与分类：

驱动：

概念：**书 P126：**

分类：嵌入式 Linux 驱动设备分为_*字符设备*_、_*块设备*_和网络设备三种类型，其中通常情况下键盘、鼠标、LCD 屏等设备驱动属于_*字符*_设备，硬盘设备属于_*块*_设备。

设备要能被使用，通常需要设备驱动软件，在 Linux 系统中，设备驱动驱动软件位于_**A**_(A.内核空间 B.用户空间)，假设有 LED 报警灯设备驱动软件 leddrv.ko，则加载该驱动软件到内核命令为_**#insmod leddrv.ko**_，测试完成后，将该软件从内核中卸载命令为_**#rmmod leddrv**_。

除网络设备外，设备驱动通常采用_文件_方式进行访问，一般包括_**打开设备**_、读写控制设备，最后_**关闭设备**_。在 Linux 中设备文件名本质上是设备号，将两者之间建立关联的命令为 mknod，假设 LED 设备驱动主设备号为 253，次设备号为 0，设备名为 leddev，则建立设备文件的具体命令为_**#mknod /dev/leddev c 253 0**_。

模块：书 P127：

内核模块是 Linux 内核向外部提供的一个插口， 其全称为动态可加载内核模块（Loadable Kernel Module， LKM），我们简称为模块。 Linux 内核之所以提供模块机制，是因为它本身是一个单内核（monolithic kernel）。单内核的最大优点是效率高，因为所有的内容都集成在一起，但其缺点是可扩展性和可维护性相对较差，模块机制就是为了弥补这一缺陷。

模块是具有独立功能的程序（本质），它可以被单独编译，但不能独立运行。它在运行时被链接到内核作为内核的一部分在内核空间运行，这与运行在用户空间的进程是不同的。模块通常由一组函数和数据结构组成，用来实现一种文件系统、一个驱动程序或其他内核上层的功能。

测试题：嵌入式 Linux 内核是可裁剪系统，通常情况下使用 **模块** 机制进行设计。模块程序和应用程序分别位于 Linux 系统的 **内核** 空间和 **用户** 空间。

嵌入式 Linux 系统模块程序 hello.c 被编译成模块 hello.ko 后，动态插入内核的命令为 **#insmod hello.ko**，查看 Linux 内核中有哪些模块的命令为 **#lsmod** ，删除内核中模块 hello.ko 的命令为 **#rmmod hello** 。

简述看门狗驱动编写的基本步骤：

编写硬件驱动代码，包括：

- **编写硬件接口函数**
- **建立文件系统与硬件接口函数的关联**
- **注册字符设备**
- **添加模块代码**

编译设备驱动并加载到内核，包括：

- **编写Makefile**
- **编译**
- **加载**
- **创建设备节点**
- **编写应用程序访问底层设备驱动**

关于 Makefile：

Linux 中自动生成 makefile 的工具集名称为 ***autotools*** ；工具集包括 aclocal、autoscan、autoconf、autoheader 和 ***automake*** 等几部分组成。

一个源码文件 hello.c 利用自动生成 makefile 的工具集后得到文件一批文件后，利用命令 ***#./configure*** 生成 Makefile，利用命令 ***#make dist*** 生成发行压缩包文件 hello.tar.gz。

嵌入式 Linux 应用程序移植常用 configure 命令生成 Makefile,一般来说，指定安装目录为/opt/soft 的命令为 ***./configure --prefix=/opt/soft*** ；如果待生成的可执行目标板为 ARM 处理器，则命令为 ***./configure --prefix=/opt/soft --HOST=arm-linux*** 。

内核完整过程：

编写一模块，向内核中添加两整数相加功能函数和两整数相减功能函数，并要求在加载模块时打印出“hello，I am in kernel now!”，卸载模块时打印“hello, I will leave from kernel now!”,编写 Makefile，并简述加载到内核和卸载出内核的基本命令。

（1）模块文件

（右图）

（2）Makefile

```
obj-m:=hello.ko
all:
make -C  /opt/linux-2.6.38  SUBDIRS=$(shell pwd)  modules
clean:
rm  -rf *.ko *.o
```

（3）编译和加载

```
#make
#inmsod hello.ko
```

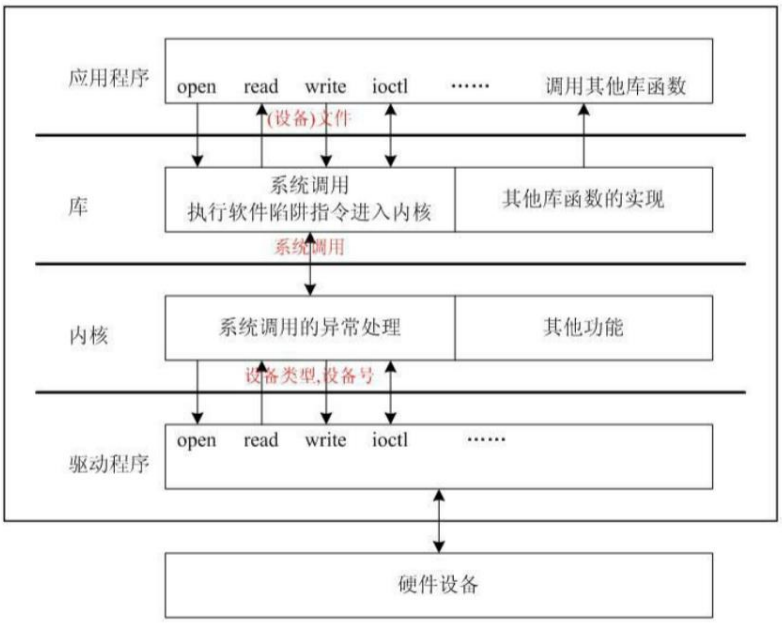
驱动与一般应用程序的区别：

驱动程序与应用程序的区别

1、应用程序以 main 开始，驱动程序没有 main，它以一个模块初始化函数作为入口。

2、应用程序从头到尾执行一个任务，驱动程序完成初始化之后不再运行，等待系统调用。

3、应用程序可以使用 GLIBC 等标准 C 函数库，驱动程序不能使用标准 C 库。



Linux 驱动程序功能

驱动程序的一般功能是：对设备初始化和释放、把数据从内核传送到硬件和从硬件读取数据、读取应用程序传送给设备文件的数据和回送应用程序请求的数据、检测和处理设备出现的错误等。

2. 设备框架、函数：

3. LED、看门狗见实验。