一、回顾：

U-Boot的作用：启动内核

内核的作用：启动应用程序

应用程序的作用：读写文件、

硬件操作：点灯、获取按键的值...

二、应用程序实现其功能的过程

1.应用程序实现这些功能的过程

应用程序能够实现其功能是通过操作底层的硬件来实现的。该过程可以描述如下：

应用程序使用类似***read() write() open()*** 等**接口函数**来调用**驱动程序**，从而操作底层硬件。

一种简单的对应关系如下图：

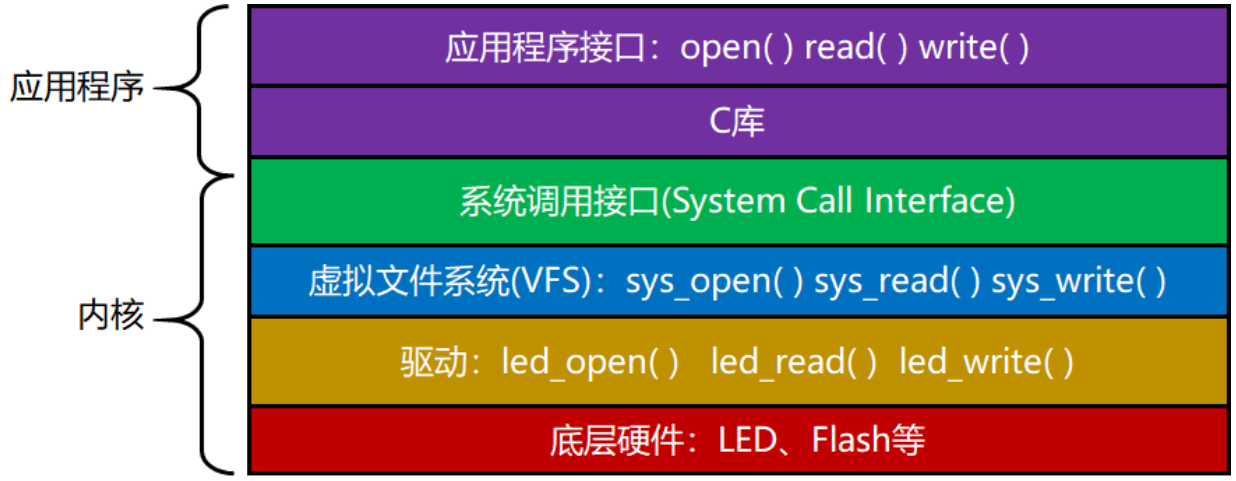


图 1 应用程序实现的框架

2.具体实现的步骤：

以一个应用程序为例

```c
int main
{
 int fd1, fd2;
 int val = 1;
 /* 1.打开LED的程序，然后写入一个值 */
 fd1 = open("/dev/led", O_RDWR);//O_RDWR表示可读可写
 write(fd1, &val, 4);
 /* 2.打开一个文本文件，然后写入一个值 */
```

```
9   fd2 = open("hello.text", O_RDWR);
10   write(fd2, &val, 4);
11  }
```

其中的***open()***、***read()***等函数是由**C库**实现的，它也属于应用程序的一部分。在应用程序调用这些函数时，C库会进入操作系统内核。C库在这个过程中会执行一条汇编指令 **swi val** 其中指令中的val表示某个值。

这条指令会引发**异常**（类似中断），在异常发生后会进入内核的**异常处理函数。**

系统调用接口（System Call Interface）会根据不同的异常原因（即不同的val值）调用不同的**异常处理函数。**

eg: 调用***open()***、***read()***及***write()***函数时分别对应值val1、val2及val3，此时**系统调用接口**会根据这些不同的值调用***sys_open***、***sys_read***及***sys_write***等函数。

这些函数包含在**VFS**(Virtual File System，**虚拟文件系统**)中，它们会根据应用程序所调用函数的**参数**来决定调用哪种设备的驱动程序。即使是调用同一种App函数，在参数不同时，所调用的设备驱动程序也不同。通常该过程是在驱动程序中调用函数来实现功能。

相同的App函数，不同的驱动程序：

eg：在上面的程序中为了实现点灯和写文件的功能都用到了***open()***、***read()***两个函数。但是在点灯和写文件的功能中***open()***函数的参数不相同，因此在VFS层中的sys_open会分别调用**不同的驱动程序**(点灯会调用LED的驱动led_open,写文件会调用Flash的驱动flash_open)

以上就是整个驱动程序的框架。

三、总结——整个程序的框架：

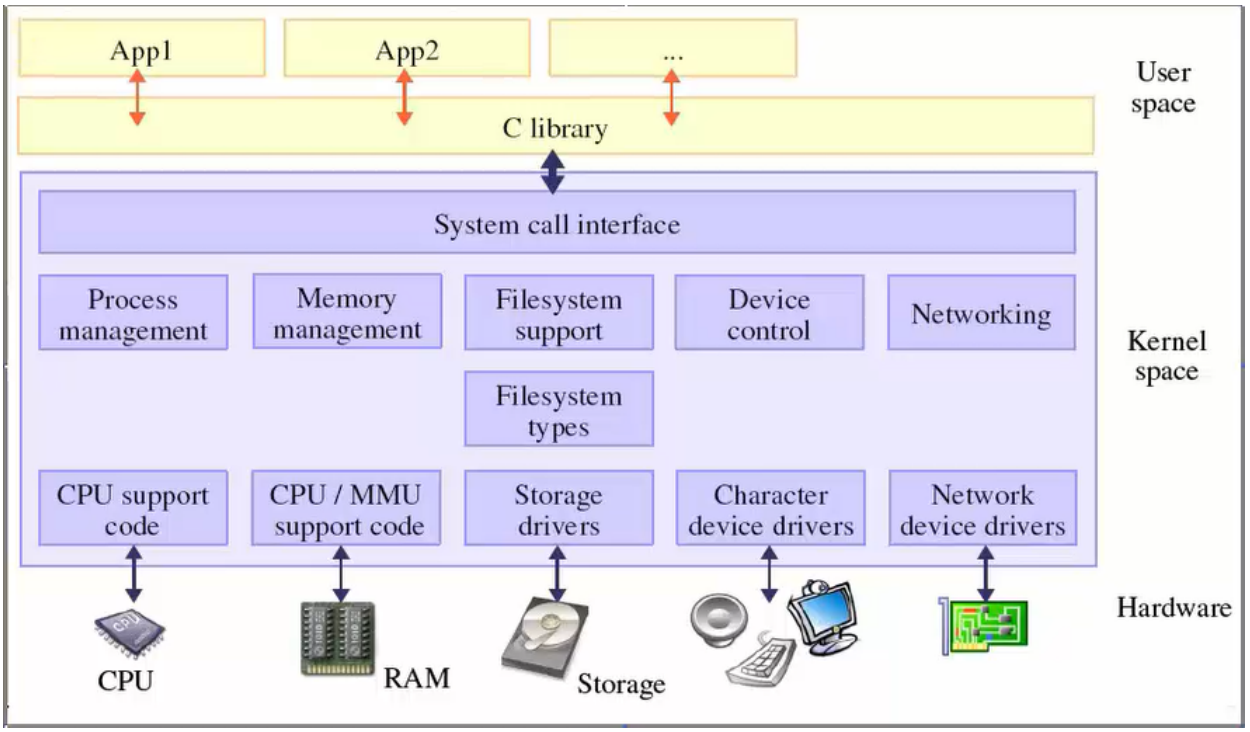


图 2 应用程序框架图

应用程序操作硬件所需要的步骤：

1.应用程序调用包含在C库中的接口函数（App调用open、read、write等函数)

2.C库通过汇编指令swi val来触发异常，不同的val值对应不同的异常

3.触发异常之后就会进入内核空间(Kernel space)

4.接着异常处理函数会取调用VFS层中的sys_open、sys_read、sys_write等函数

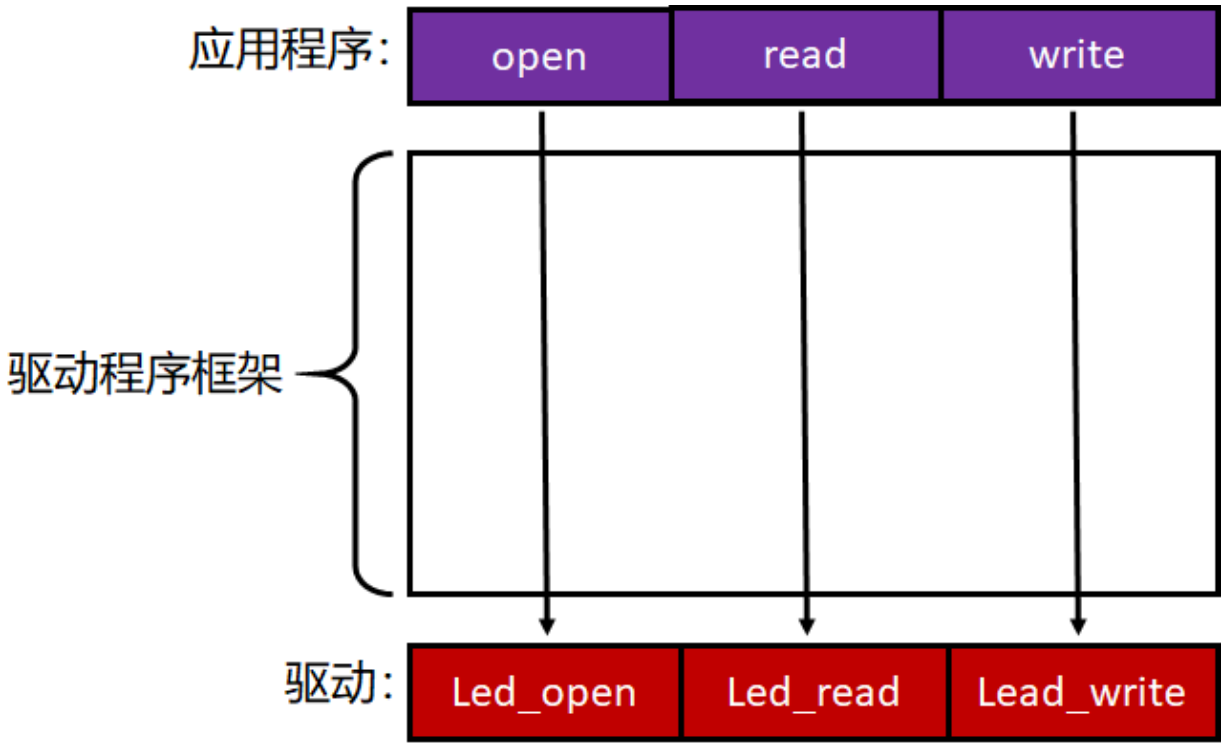
5.VFS中的这些函数会根据APP函数中打开不同的文件的不同属性来找到更底层的驱动程序

四、结语：

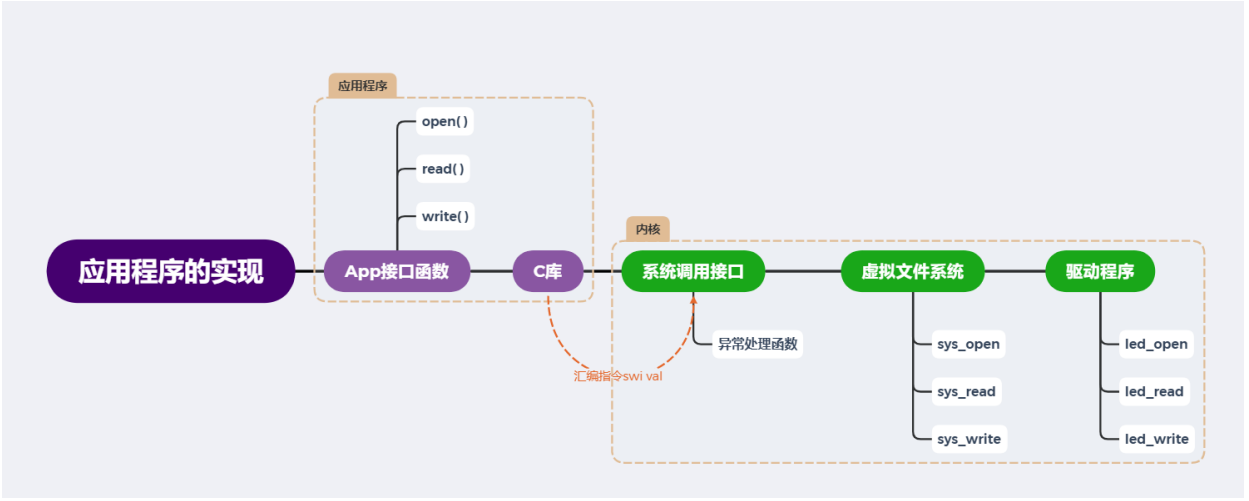
在应用程序中我们能够通过一些函数接口(例如：***open***、***read***及***write***)来操作硬件。是因为有与这些函数接口相对应的驱动程序(eg:led_open、led_read及led_write)

那么应用程序中的函数是如何与底层的驱动程序之间对应的?

----依赖于**驱动程序的框架**来实现


应用程序:
open
read
write
驱动程序框架
驱动:
Led_open
Led_read
Lead_write

六、思维导图:


应用程序
open()
read()
write()
应用程序的实现
App接口函数
C库
内核
系统调用接口
虚拟文件系统
驱动程序
异常处理函数
汇编指令swi val
sys_open
sys_read
sys_write
led_open
led_read
led_write