基于 ARM 处理器的 Semihosting 实现

作者：
李明
清华大学智能技术与系统国家重点实验室

摘要：
Semihosting（半主机）是当前在基于 ARM 的嵌入式开发中普遍使用的一种调试手段，它是在目标系统和调试器之间进行输入 / 输出的重要机制，对用户程序能否使用标准 C 库进行开发起非常重要的作用。这篇文章从它的运行机制和实现原理上做了深入的分析，并给出一个具体的实例，说明了如何通过实现其相应的软中断接口，使得目标系统能够支持 Semihosting 的请求，从而在目标系统上实现相关的标准库支持。最后对于 Semihosting 和 Angel 的关系做了简要总结。

一、Semihosting 机制

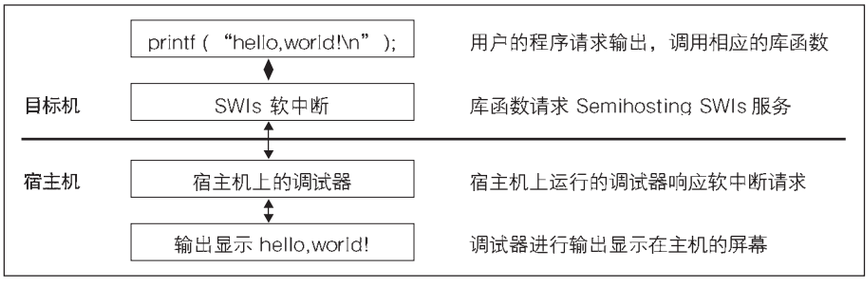
Semihosting 是在 ARM 处理器平台上进行嵌入式开发和调试时使用的一种输入 / 输出的机制，表现为目标机上的应用程序向运行在宿主机上的调试器发出Semihosting请求实现I/O功能。举例来说，用户的程序可以使用C库中的函数调用，例如printf、scanf等来通过开发主机进行输入/输出，而不一定要求目标平台上具有键盘或显示器等设备。

这种机制在嵌入式开发初期是非常有用的，因为开始的时候目标平台还不能提供输入/输出的功能，那么在宿主机上进行调试，就需要由宿主机来提供这些功能，以便可以和用户程序实现交互。一旦用户程序调试通过后，就可以由目标平台来实现 Semihosting 的功能，这样程序就可以直接在目标机上运行了。

二、原理分析

1） 运行机制

Semihosting 的功能实际上是通过 ARM 的软中断（SWI）机制来实现的。用户的程序在使用相关的输入输出函数时，会触发一个相应的软件中断。此时调试器来响应这个软中断并做出相应处理。这样运行在目标机上的用户程序就可以和宿主机上的调试器来进行交互，而不是由目标机来进行输入输出的处理。以 printf 为例，如图所示：



▲ 图 1.Semihosting 实现原理图示

2） 三种调试代理

从上面的原理图可以看出，实际上 SWI 就类似于一种系统调用的服务入口，用户输入输出库函数调用，通过软中断的方式提出 Semihosting 请求，由宿主机上的调试器响应并进行服务。所有 ARM 提供的调试代理（debug agent），例如 ARMulator、Angel 和 Multi-ICE 等都使用统一的 SWI 接口进行服务，因此对于用户而言，使用任何一种调试手段都不用对上层的代码做修改，移植起来很方便。

ARMulator 是一种软件仿真手段，它无需目标平台，直接可以在开发主机上进行仿真实现。当 Semihosting SWI 指令执行的时候，ARMulator 截获这个中断请求，进行相应的系统服务。此时真正的 SWI 的服务程序就不起作用了。当然，也可以在 ARMulator 中将 Semihosting 的支持功能关闭。

Angel 是驻留在目标平台上的监控模块，在目标平台加电后进行初始化，并安装 SWI 的处理程序，一旦目标平台上运行的程序执行一条 Semihosting 的软中断指令，Angel SWI handler 就会进行响应并和主机上的调试器进行交互。

Multi-ICE 是使用 JTAG 接口协议的在线仿真器。它采用在 SWI 服务入口

设置断点的办法，当SWI发生时 Multi-ICE 检查SWI number ，如果是属于Semihosting 的调用号，则Multi-ICE进行服务并继续运行用户程序，否则，Multi-ICE 就停止处理器，并报告一个错误信息。

3） 软中断的接口

在Semihosting的实现中，需要使用一个特殊的软中断号来标识，这个中断号一般是固定的，在ARM 状态下为0x123456，在 Thumb 状态下则为 0xAB。处理器通过执行SWI 0x123456这条指令来请求 Semihosting 服务。

在进入Semihosting的服务处理程序后，还需要有一个标识不同 Semihosting 功能调用的操作号(operation type)，这个操作号通常存放在 r0 寄存器中。例如SYS_WRITE 的操作号是 0x05，在执行SWI 0x123456 之前，需要将 0x05 传给r0 寄存器，同时SYS_WRITE 调用需要的参数会通过一个结构指针传给r1寄存器，这样在 SWI 发生后就可以根据r0和r1 寄存器中的操作号和传递的参数来进行相应服务。

下面的这张表中列出了一些比较重要的 Semihosting 操作号和相关的系统调用：

三、Semihosting 的实现

当使用Multi-ICE或ARMulator等调试代理在调试器中完成了程序测试之后，将代码下载到开发板上的时候，如果开发板上已经有了Angel等驻留监控模块，那么Semihosting的功能服务就通过Angel来实现，否则就需要自己实现 Semihosting SWIs 的软中断服务。通常我们都会给自己的目标板上移植一个专门用于下载用户程序的bootloader，例如 BootStrap Loader、armboot、redboot等，但这些bootloader一般都不会实现 Semihosting 的功能。

这里我们以输出helloworld的程序为例，说明如何在已有bootloader的目标系统上通过实现 Semihosting SWIs 来提供 Semihosting 功能。

1） 编译获得目标代码

```
#include <stdio.h>
int main( void )
{
    printf ( "hello, world! \n" );
    return 1;
}
```

一个简单的用户输出程序如上所示，采用ARM 公司的 armcc 编译器和armlink 连接器编译连接后，目标代码将会调用到如下的一些 Semihosting SWIs：

1. SYS_HEAPINFO （0x16）
   用来初始化用户程序的堆栈空间指针
2. SYS_OPEN （0x01）
   用来打开stdin、stdout、stderror
3. SYS_ISTTY （0x09）
   用来判断打开文件句柄是否是一个无缓冲的终端
4. SYS_WRITE （0x05）
   用来将指定位置和长度的字符串写入相关文件
5. SYS_CLOSE （0x02）
   用来关闭stdin、stdout、stderror
6. angel_SWIreason_Report Exception （0x18）
   用来退出（Exit）主程序

2） Semihosting SWIs 发生前的寄存器分配

以 _sys_write 为例，在进行SWI 0x123456 调用之前，在 r0 寄存器中已经保存了SYS_WRITE 的Semihosting 的操作号 0x05，在 r1 寄

ID	Semihosting Operation Type	System Calls
0x01	SYS_OPEN	int _sys_read(FILEHANDLE fh, unsigned char *buf, unsigned len, int mode)
0x02	SYS_CLOSE	int _sys_close(FILEHANDLE fh)
0x03	SYS_WRITEC	
0x04	SYS_WRITE0	
0x05	SYS_WRITE	int _sys_write(FILEHANDLE fh, const unsigned char *buf, unsigned len, int mode)
0x06	SYS_READ	int _sys_read(FILEHANDLE fh, unsigned char *buf, unsigned len, int mode)
0x07	SYS_READC	
0x08	SYS_ISERROR	
0x09	SYS_ISTTY	int _sys_istty(FILE *f)
0x0A	SYS_SEEK	int _sys_seek(FILEHANDLE fh, long pos)
0x0C	SYS_FLEN	long _sys_flen(FILEHANDLE fh)
0x13	SYS_ERRNO	
0x16	SYS_HEAPINFO	__value_in_regs struct __initial_stackheap __user_initial_stackheap (unsignedR0, unsigned SP, unsigned R2, unsigned SL)
0x18	angel_SWIreason_ReportException	

存器中保存了一个指向该系统调用sys_write的参数列表数据结构的指针，这些参数实际上就保存在该函数调用的栈空间中，传递的指针也是放在参数之后的栈空间上。这里的参数包括一个相关的文件句柄，输出字符串的指针，需要输出字符串的长度。

int _sys_write (FILEHANDLE fh, const unsigned char *buf, unsigned len, int mode)

汇编后的 sys_write 实现代码：

```
_sys_write
    STR     r14,[r13,#-4]!
    SUB     r13,r13,#0x14
    STMIB   r13,{r0-r2}
    ADD     r1,r13,#4
    MOV     r0,#5
    SWI     0x123456
    ADD     r13,r13,#0x14
    LDR     pc,[r13],#4
```

_sys_write 实现流程说明：

1> 保存 r14 返回地址到当前栈顶

2> 调整栈指针，到往下 5 个字的位置

3> 使用预先增加装载的方式，将 r0-r2 三个寄存器的内容装载到如下图位置

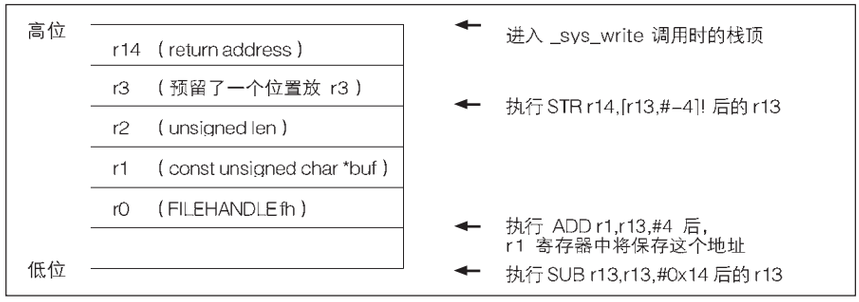
4> 保存 sys_write 调用参数的数据块地址到 r1 寄存器

5> 保存sys_write 调用的Semihosting 操作号 0x05 到 r0 寄存器

6> 使用软中断SWI 来请求Semihosting 服务

7> 恢复 r13 的值到原先保存的返回地址处

8> 读取返回地址到 pc 中，实现跳转；同时更新栈指针到函数调用前的位置



▲ 图 2 _sys_write 调用图示说明

其他的系统调用的实现与_sys_write很类似，基本上都是使用r0来保存操作号，r1来保存传递参数数据块的指针，然后进行 SWI 0x123456 来请求 Semihosting 服务。

3） SWI 中断服务程序的实现

在发生 SWI 软中断后，ARM 处理器会自动执行异常向量表中（即 0x08 的位置）的指令，通常这里需要放置一条跳转指令，使得 pc 跳转到 SWI handler 的入口。一般进行SWI 服务程序的前面部分需要获得 SWI 的中断号，这个中断号是在 SWI 指令机器码的后 24 位。一个比较常见的 SWI_Handler 如下所示：

```
SWI_Handler
    STMFD   sp!, {r0-r1}        保存 r0、r1 寄存器
    LDR     r0,[lr,#-4]         获得 SWI 指令的机器码到 r0
    BIC     r0,r0,#0xff000000   得到 SWI 指令后 24 位的中断号
    LDR     r1, =0x123456       0x123456 是 Semihosting 的中断号
    CMP     r0, r1              判断是否是请求 Semihosting 服务
    LDMFD   sp!, {r0-r2}        恢复 r0、r1 寄存器
    BNE     Exit_SWI            如果非，则退出
C_Handler
    STMFD   sp!, {lr}           保存返回地址
    BL      C_SWI_Handler       这里开始进行 Semihosting SWIs 服务
    LDMFD   sp!, {pc}^          从栈中恢复，从返回地址处继续执行
Exit_SWI
    MOVS    pc, r14             从返回地址处继续执行
```

4） 实现相关的 Semihosting SWIs

上面使用 main 入口的 helloworld 程序，使用 ARM 的编译器armcc 和连接器 armlink 生成的目标代码，会自动连接负责初始化堆栈和C库的代码，其工作流程如下：

a）将执行文件中的 RO 段和 RW 段从 load address 复制到 execution address

b）初始化 ZI 区域，用0来初始化变量

c）跳转到__rt_entry执行如下4个调用：

i） 调用 __rt_statckheap_init，建立程序的堆和栈

ii） 调用 __rt_lib_init，初始化程序用到的 C 库，并为 main 传递参数

iii）调用 main，即用户程序的入口

iv）调用exit，即退出，中止程序执行

在 helloworld 程序的运行过程中，将会在如下表中说明的函数调用链里使用到 Semihosting SWIs 服务，如果要使目标板支持Semihosting，保证已经调试成功的程序可以直接移植到开发板上，就需要在目标机上实现这些 Semihosting SWIs 服务。

函数调用链	描　述	Semihosting SWI	操作号
__rt_statckheap_init-> __user_initial_stackheap	初始化程序的堆栈空间	SYS_HEAPINFO	0x16
__rt_lib_init -> _initio -> freopen -> _sys_open	打开标准输入 stdin、输出 stdout 和错误 stderr 设备	SYS_OPEN	0x01
main -> _printf -> __vfprintf -> fputc -> __flsbuf -> _writebuf -> _sys_write	向标准输出字符串	SYS_WRITE	0x05
main -> _printf -> __vfprintf -> fputc -> __flsbuf -> _sys_istty	用来判断是否是不缓存的终端	SYS_ISTTY	0x09
exit -> __32__rt_exit -> __rt_lib_shutdown -> _terminateio -> fclose -> _sys_close	关闭标准输入 stdin、输出 stdout 和错误 stderr 设备	SYS_CLOSE	0x02
exit -> __32__rt_exit -> __rt_abort1 -> _sys_exit	程序结束，退出执行	angel_SWIreason_ReportException	0x18

四、总结

Semihosting 对于 ARM 处理器来说，它的实现是和 Angel 监控模块密切相关的，准确的说，它是属于 Angel 工作功能的一部分。而它对于目标系统能否支持上层应用的开发使用 ANSI C 库也是不可或缺的，因为ARM C的标准库中有相当一部分系统调用都是和 Semihosting SWI 相连接的。因此分析了 Semihosting 的工作原理，对于我们了解基于 ARM 的嵌入式程序开发都是有相当帮助的。这里的介绍只是目前探索工作的一个总结，更多的资料可以从 ARM 公司提供的文档中找到。

参考文献：

[1] ARM Ltd. ARM Developer Suite Debug Target Guide. http://www.arm.com/. 1999-2001

[2] ARM Ltd. ARM Software Development Toolkit. http://www.arm.com/. 1998