专题5-从MACHINE_START开始

注：下面的内容是以linux-2.6.38和mini6410为例进行学习的。

玩过或者移植过arm-linux的都应该知道在/arch/arm目录下有许多与具体处理器相关的目录，当然对于6410的话所对应的目录就是mach-s3c64xx，在里面找到与具体板子相关的文件mach-mini6410.c，没错，就是它。无论是出于想移植到新的内核还是出于想深入学习某一款arm等，对这个文件的学习是必不可少的。这个文件大部分内容是对平台设备（例如串口，LCD，Nand falsh等）的结构体初始化，在这个文件的最后有一个非常重要的宏：

```
1 MACHINE_START(MINI6410, "MINI6410")
2         /* Maintainer: Ben Dooks <ben-linux@fluff.org> */
3         .boot_params    = S3C64XX_PA_SDRAM + 0x100,
4
5         .init_irq       = s3c6410_init_irq,
6         .map_io         = mini6410_map_io,
7         .init_machine   = mini6410_machine_init,
8         .timer          = &s3c24xx_timer,
9 MACHINE_END
```

MINI6410这个宏在/arch/arm/tools/mach-types文件里定义：

```
mini6410        MACH_MINI6410        MINI6410        2520
```

MACHINE_START的定义在arch/arm/include/asm/mach/arch.h，如下：

```
1 #define MACHINE_START(_type,_name)                      \
2 static const struct machine_desc __mach_desc_##_type    \
3  __used                                                 \
4  __attribute__((__section__(".arch.info.init"))) = {    \
5         .nr             = MACH_TYPE_##_type,            \
6         .name           = _name,
7
8 #define MACHINE_END                             \
9 };
```

噢，其实就是定义了一个struct machine_desc类型结构体变量，这个结构体还定义了其他一些成员，接下来着重关注.init_machine这个成员，它是一个函数指针，值为mini6410_machine_init，这个函数也在mach-mini6410.c中定义。内容是什么呢？呵呵，因为在这里只给出大体流程，具体内容先不分析。现在最关心的是这个结构体变量在哪里被调用，从而调用它里面的成员和成员函数呢？先来看/arch/arm/kernel/setup.c里面的setup_arch()函数：

```
 1 void __init setup_arch(char **cmdline_p)
 2 {
 3         struct tag *tags = (struct tag *)&init_tags;
 4         struct machine_desc *mdesc;
 5         char *from = default_command_line;
 6
 7         unwind_init();
 8
 9         setup_processor();
10         mdesc = setup_machine(machine_arch_type);
11         machine_desc = mdesc;
12         machine_name = mdesc->name;
.....................
```

这个函数在/init/main.c的start_kernel()函数里被调用。看到第10行，这里的setup_machine()函数的作用就是找到我们想要的struct machine_desc类型的变量，也就是在mach-mini6410.c里定义那个变量。函数的参数machine_arch_type的值是什么呢？继续看：

```
 1 #ifdef CONFIG_MACH_MINI6410
 2 # ifdef machine_arch_type
 3 #  undef machine_arch_type
 4 #  define machine_arch_type     __machine_arch_type
 5 # else
 6 #  define machine_arch_type     MACH_TYPE_MINI6410
 7 # endif
 8 # define machine_is_mini6410()  (machine_arch_type == MACH_TYPE_MINI6410)
 9 #else
10 # define machine_is_mini6410()  (0)
11 #endif
```

第6行，MACH_TYPE_MINI6410宏为：

```
#define MACH_TYPE_MINI6410              2520
```

也就是说参数machine_arch_type的值为2520。在setup_machine()函数里主要调用了lookup_machine_type()函数来查找对应的type，应该是出于效率的原因，这个函数是通过汇编实现的，在此就不给出具体代码了。

到这里，知道了在/init/main.c的start_kernel()函数里调用了setup_arch()，在setup_arch()里找到了具体的struct machine_desc类型的变量，但是在哪里通过这个变量调用里面的成员或成员函数的呢？继续找。还是在setup.c里，看到了这样一个函数：

```
1 static int __init customize_machine(void)
2 {
3         /* customizes platform devices, or adds new ones */
4         if (machine_desc->init_machine)
```

```
5                 machine_desc->init_machine();
6         return 0;
7 }
8 arch_initcall(customize_machine);
```

终于看到了，成员函数init_machine就是在这里被调用的。但是它没有被显式调用，而是放在了arch_initcall这个宏里，去看看它怎么定义的：

```
#define arch_initcall(fn)               __define_initcall("3",fn,3)
```

再看__define_initcall宏：

```
1 #define __define_initcall(level,fn,id) \
2         static initcall_t __initcall_##fn##id __used \
3         __attribute__((__section__(".initcall" level ".init"))) = fn
```

嗯，它被链接到了.initcall段里，现在简单看看/arch/arm/kernel/vmlinux.lds这个链接脚本里关于initcall的定义：

```
 1 __initcall_start = .;
 2 *(.initcallearly.init) __early_initcall_end = .;
 3 *(.initcall0.init) *(.initcall0s.init)
 4 *(.initcall1.init) *(.initcall1s.init)
 5 *(.initcall2.init) *(.initcall2s.init)
 6 *(.initcall3.init) *(.initcall3s.init)
 7 *(.initcall4.init) *(.initcall4s.init)
 8 *(.initcall5.init) *(.initcall5s.init)
 9 *(.initcallrootfs.init)
10 *(.initcall6.init) *(.initcall6s.init)
11 *(.initcall7.init) *(.initcall7s.init)
12 __initcall_end = .;
```

可以看到customize_machine()被放到了.initcall3.init里。说了那么多定义，究竟它在哪里被调用啊？好吧，它是在/init/main.c里一个叫do_initcalls()的函数里被调用，去看看呗：

```
1 extern initcall_t __initcall_start[], __initcall_end[], __early_initcall_end[];
2
3 static void __init do_initcalls(void)
4 {
5         initcall_t *fn;
6
7         for (fn = __early_initcall_end; fn < __initcall_end; fn++)
8                 do_one_initcall(*fn);
9 }
```

看到第1行，很熟悉吧。在for循环里依次调用了从__early_initcall_end开始到__initcall_end结束的所有函数。customize_machine()也是在其间被调用。

好了，到这里差不多该结束了，最后总结一下这些函数调用顺序：

start_kernel()--->setup_arch()--->do_initcalls()--->customize_machine()--->mini6410_machine_init()