in Documentation/kernel-parameters.txt

```
    boot_delay=    Milliseconds to delay each printk during boot.
                   Values larger than 10 seconds (10000) are changed to
                   no delay (0).
                   Format: integer
```

在boot阶段每次调用printk()都要延迟上boot_delay毫秒，why？没想明白动机?!

一种可能，使得console driver有足够时间输出log。如果在boot阶段，突然crash,甚至通过printk()输出的log都没有机会真正输出。

因为真正输出是依赖于console driver的输出，比如uart的输出。这里每次printk()加入delay，就给了console尽量输出log的机会。

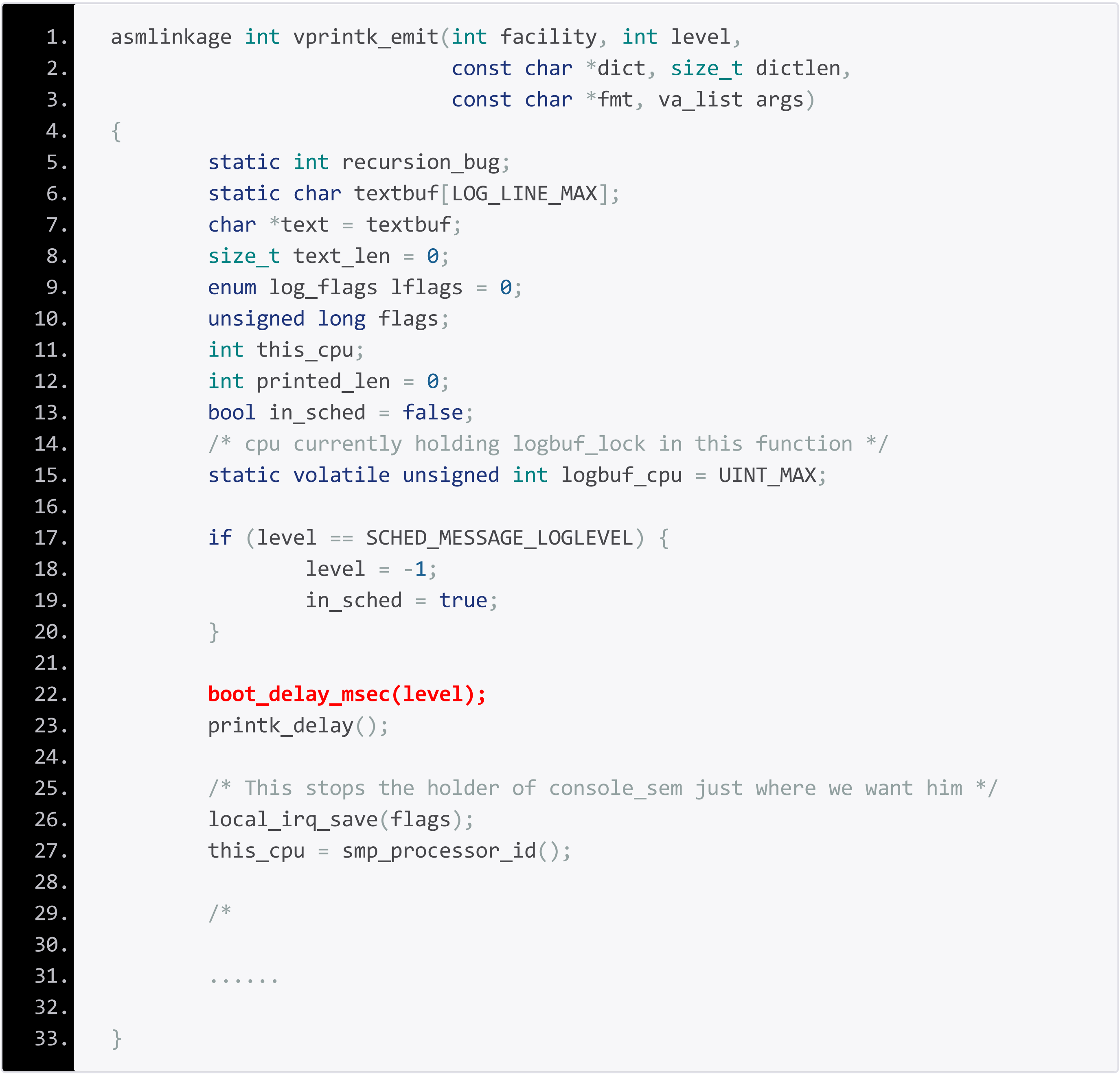
这只是我的guess!

in kernel/printk/printk.c

```
printk()
    |
      |
 \|/
vprintk_emit()
      |
      |
```

\|/

boot_delay_msec(level);



```c
asmlinkage int vprintk_emit(int facility, int level,
                            const char *dict, size_t dictlen,
                            const char *fmt, va_list args)
{
        static int recursion_bug;
        static char textbuf[LOG_LINE_MAX];
        char *text = textbuf;
        size_t text_len = 0;
        enum log_flags lflags = 0;
        unsigned long flags;
        int this_cpu;
        int printed_len = 0;
        bool in_sched = false;
        /* cpu currently holding logbuf_lock in this function */
        static volatile unsigned int logbuf_cpu = UINT_MAX;

        if (level == SCHED_MESSAGE_LOGLEVEL) {
                level = -1;
                in_sched = true;
        }

        boot_delay_msec(level);
        printk_delay();

        /* This stops the holder of console_sem just where we want him */
        local_irq_save(flags);
        this_cpu = smp_processor_id();

        /*

        ......

}
```

```
static int boot_delay; /* msecs delay after each printk during bootup */
static unsigned long long loops_per_msec;       /* based on boot_delay */

static int __init boot_delay_setup(char *str)
{
        unsigned long lpj;

        lpj = preset_lpj ? preset_lpj : 1000000;        /* some guess */
        loops_per_msec = (unsigned long long)lpj / 1000 * HZ;

        get_option(&str, &boot_delay);
        if (boot_delay > 10 * 1000)
                boot_delay = 0;

        pr_debug("boot_delay: %u, preset_lpj: %ld, lpj: %lu, "
                "HZ: %d, loops_per_msec: %llu\n",
                boot_delay, preset_lpj, lpj, HZ, loops_per_msec);
        return 0;
}
early_param("boot_delay", boot_delay_setup);

static void boot_delay_msec(int level)
{
        unsigned long long k;
        unsigned long timeout;

        if ((boot_delay == 0 || system_state != SYSTEM_BOOTING)
 ①
                || (level >= console_loglevel && !ignore_loglevel)) {
                return;
        }

        k = (unsigned long long)loops_per_msec * boot_delay;
        ②

        timeout = jiffies + msecs_to_jiffies(boot_delay);
        while (k) {

                                    ③
                k--;
                cpu_relax();

                           ④
                /*
                 * use (volatile) jiffies to prevent
                 * compiler reduction; loop termination via jiffies
                 * is secondary and may or may not happen.
                 */
                if (time_after(jiffies, timeout))
                        break;
                touch_nmi_watchdog();

  ⑤
```

```
        }
}
```

①

只有在boot阶段才boot_delay的效果才有效

②

通过loops_per_msec来估计要循环多少次。loops_per_msec本身就是一个估计值，在calibrate_delay()中估算。

③

纯粹浪费CPU的循环

④

内存屏障，无聊的浪费时间的时候使得可能乱序(out of order)执行的instruction同步。

```
in arch/arm/include/asm/processor.h

#if __LINUX_ARM_ARCH__ == 6 || defined(CONFIG_ARM_ERRATA_754327)
#define cpu_relax()                     smp_mb()
#else
#define cpu_relax()                     barrier()
#endif
```

⑤

应该是为了防止在固定间隔内没有“喂” watchdog而可能重启的逻辑。