# printk函数分析

**[asmlinkage](https://elixir.bootlin.com/linux/1.0/C/ident/asmlinkage)** int **[printk](https://elixir.bootlin.com/linux/1.0/C/ident/printk)**(const char \***[fmt](https://elixir.bootlin.com/linux/1.0/C/ident/fmt)**, ...)

{

**[va\_list](https://elixir.bootlin.com/linux/1.0/C/ident/va_list)** **[args](https://elixir.bootlin.com/linux/1.0/C/ident/args)**;

int i;//保持不变

char \*msg, \*p, \***[buf\_end](https://elixir.bootlin.com/linux/1.0/C/ident/buf_end)**;//msg在每行输出日志级别后

static char **[msg\_level](https://elixir.bootlin.com/linux/1.0/C/ident/msg_level)** = -1;//在每行开始设置，在结束置-1

long flags;

**[save\_flags](https://elixir.bootlin.com/linux/1.0/C/ident/save_flags)**(flags);//存储状态寄存器

**[cli](https://elixir.bootlin.com/linux/1.0/C/ident/cli)**();//输入指令

**[va\_start](https://elixir.bootlin.com/linux/1.0/C/ident/va_start)**(**[args](https://elixir.bootlin.com/linux/1.0/C/ident/args)**, **[fmt](https://elixir.bootlin.com/linux/1.0/C/ident/fmt)**);

//**将fmt后的参数信息保存到args中，va\_start宏的作用就是将指针 args跳过fmt参数，指向第一个要解析的可变参数。**

i = **[vsprintf](https://elixir.bootlin.com/linux/1.0/C/ident/vsprintf)**(buf + 3, **[fmt](https://elixir.bootlin.com/linux/1.0/C/ident/fmt)**, **[args](https://elixir.bootlin.com/linux/1.0/C/ident/args)**); */\* hopefully i < sizeof(buf)-4 \*/*

//**通过args获得可变参数列表**

**//根据解析fmt中控制字符，比如%d，%s等，将args指向位置的 参数转换成字符放入buf+3中**

**//更新args，重复第二步，直到全部解析完毕为止**

相当于把数据格式化存入buf+3

**[buf\_end](https://elixir.bootlin.com/linux/1.0/C/ident/buf_end)** = buf + 3 + i;

**[va\_end](https://elixir.bootlin.com/linux/1.0/C/ident/va_end)**(**[args](https://elixir.bootlin.com/linux/1.0/C/ident/args)**);

//**将args参数设为0**

for (p = buf + 3; p < **[buf\_end](https://elixir.bootlin.com/linux/1.0/C/ident/buf_end)**; p++) {

msg = p;

if (**[msg\_level](https://elixir.bootlin.com/linux/1.0/C/ident/msg_level)** < 0) {

//**没有设置日志级别，添加日志级别**

if (

p[0] != '<' ||

p[1] < '0' ||

p[1] > '7' ||

p[2] != ‘>’

//**设置了错误的级别格式**

) {

p -= 3;

p[0] = '<';

p[1] = **[DEFAULT\_MESSAGE\_LOGLEVEL](https://elixir.bootlin.com/linux/1.0/C/ident/DEFAULT_MESSAGE_LOGLEVEL)** - 1 + '0';

p[2] = '>';

} else

msg += 3;

**[msg\_level](https://elixir.bootlin.com/linux/1.0/C/ident/msg_level)** = p[1] - '0';

}

for (; p < **[buf\_end](https://elixir.bootlin.com/linux/1.0/C/ident/buf_end)**; p++) {

//**把字符p存到log\_buf缓冲区中，缓冲区满了就会覆盖开始的数据**

**[log\_buf](https://elixir.bootlin.com/linux/1.0/C/ident/log_buf)**[(**[log\_start](https://elixir.bootlin.com/linux/1.0/C/ident/log_start)**+**[log\_size](https://elixir.bootlin.com/linux/1.0/C/ident/log_size)**) & (**[LOG\_BUF\_LEN](https://elixir.bootlin.com/linux/1.0/C/ident/LOG_BUF_LEN)**-1)] = \*p;

if (**[log\_size](https://elixir.bootlin.com/linux/1.0/C/ident/log_size)** < **[LOG\_BUF\_LEN](https://elixir.bootlin.com/linux/1.0/C/ident/LOG_BUF_LEN)**)

**[log\_size](https://elixir.bootlin.com/linux/1.0/C/ident/log_size)**++;

else

**[log\_start](https://elixir.bootlin.com/linux/1.0/C/ident/log_start)**++;

**[logged\_chars](https://elixir.bootlin.com/linux/1.0/C/ident/logged_chars)**++;

if (\*p == '\n')

**[break](https://elixir.bootlin.com/linux/1.0/C/ident/break)**;

}

if (**[msg\_level](https://elixir.bootlin.com/linux/1.0/C/ident/msg_level)** < **[console\_loglevel](https://elixir.bootlin.com/linux/1.0/C/ident/console_loglevel)** && **[console\_print\_proc](https://elixir.bootlin.com/linux/1.0/C/ident/console_print_proc)**) {

//**缓冲区循环使用就会出现(start & LOG\_BUF\_MASK) > (end & LOG\_BUF\_MASK)的清况,于是就分两部分来发送.**

char tmp = p[1];

p[1] = ‘\0’;//**截断**

(\***[console\_print\_proc](https://elixir.bootlin.com/linux/1.0/C/ident/console_print_proc)**)(msg);//**输出**

p[1] = tmp;//**恢复**

}

if (\*p == ‘\n’)//重制msglevel，每行msglevel要更新

**[msg\_level](https://elixir.bootlin.com/linux/1.0/C/ident/msg_level)** = -1;

}

**[restore\_flags](https://elixir.bootlin.com/linux/1.0/C/ident/restore_flags)**(flags);//恢复状态寄存器

**[wake\_up\_interruptible](https://elixir.bootlin.com/linux/1.0/C/ident/wake_up_interruptible)**(&**[log\_wait](https://elixir.bootlin.com/linux/1.0/C/ident/log_wait)**);//唤醒中断

return i;

}

不足之处：不支持异步（asynchronous）工作模式；在一些情况下，printk()必须要先输出到不依赖lock的safe buffer里，导致message最终被copy到真正的ring buffer的时候更新的timestamp不精确，也可能会导致message丢失，或者导致CPU异常offline的时候buffer没有被刷出去；printk()对各个级别的log一视同仁。某些频繁输出的信息如果被错误设置成urgent级别，可能会导致latency问题，从而让大家会调整level来导致丢失其他的urgent信息。