程序 14-23 linux/include/linux/kernel.h

```
1  /*
2   * 'kernel.h' contains some often-used function prototypes etc
3   */
   /*
    * 'kernel.h'定义了一些常用函数的原型等。
    */
   // 验证给定地址开始的内存块是否超限。若超限则追加内存。( kernel/fork.c, 24 )。
4  void verify_area(void * addr,int count);
   // 显示内核出错信息，然后进入死循环。( kernel/panic.c, 16 )。
   // 函数名前的关键字 volatile 用于告诉编译器 gcc 该函数不会返回。这样可让 gcc 产生更好
   // 一些的代码，更重要的是使用这个关键字可以避免产生某未初始化变量的假警告信息。
5  volatile void panic(const char * str);
   // 进程退出处理。( kernel/exit.c, 262 )。
6  volatile void do_exit(long error_code);
   // 标准打印（显示）函数。( init/main.c, 151)。
7  int printf(const char * fmt, ...);
   // 内核专用的打印信息函数，功能与 printf()相同。( kernel/printk.c, 21 )。
8  int printk(const char * fmt, ...);
   // 控制台显示函数。( kernel/chr_drv/console.c, 995）。
9  void console_print(const char * str);
   // 往 tty 上写指定长度的字符串。( kernel/chr_drv/tty_io.c, 290 )。
10 int tty_write(unsigned ch,char * buf,int count);
   // 通用内核内存分配函数。( lib/malloc.c, 117)。
11 void * malloc(unsigned int size);
   // 释放指定对象占用的内存。( lib/malloc.c, 182)。
12 void free_s(void * obj, int size);
   // 硬盘处理超时。（kernel/blk_drv/hd.c，318）。
13 extern void hd_times_out(void);
   // 停止蜂鸣。（kernel/chr_drv/console.c，944）。
14 extern void sysbeepstop(void);
   // 黑屏处理。（kernel/chr_drv/console.c，981）。
15 extern void blank_screen(void);
   // 恢复被黑屏的屏幕。（kernel/chr_drv/console.c，988）。
16 extern void unblank_screen(void);
17
18 extern int beepcount;              // 蜂鸣时间嘀嗒计数（kernel/chr_drv/console.c，988）。
19 extern int hd_timeout;             // 硬盘超时滴答值（kernel/blk_drv/blk.h）。
20 extern int blankinterval;          // 设定的屏幕黑屏间隔时间。
21 extern int blankcount;             // 黑屏时间计数（kernel/chr_drv/console.c，138、139）。
22
23 #define free(x) free_s((x), 0)
24
25 /*
26  * This is defined as a macro, but at some point this might become a
27  * real subroutine that sets a flag if it returns true (to do
28  * BSD-style accounting where the process is flagged if it uses root
29  * privs).  The implication of this is that you should do normal
30  * permissions checks first, and check suser() last.
31  */
   /*
    * 下面函数是以宏的形式定义的，但是在某方面来看它可以成为一个真正的子程序，
    * 如果返回是 true 时它将设置标志（如果使用 root 用户权限的进程设置了标志，则用
```

```
    * 于执行 BSD 方式的计帐处理）。这意味着你应该首先执行常规权限检查，最后再
    * 检测 suser()。
    */
32 #define suser() (current->euid == 0)            // 检测是否是超级用户。
33
```