程序 8-4 linux/kernel/mktime.c 程序

```
1  /*
2   *  linux/kernel/mktime.c
3   *
4   *  (C) 1991  Linus Torvalds
5   */
6
7  #include <time.h>            // 时间头文件，定义了标准时间数据结构 tm 和一些处理时间函数原型。
8
9  /*
10  * This isn't the library routine, it is only used in the kernel.
11  * as such, we don't care about years<1970 etc, but assume everything
12  * is ok. Similarly, TZ etc is happily ignored. We just do everything
13  * as easily as possible. Let's find something public for the library
14  * routines (although I think minix times is public).
15  */
16 /*
17  * PS. I hate whoever though up the year 1970 - couldn't they have gotten
18  * a leap-year instead? I also hate Gregorius, pope or no. I'm grumpy.
19  */
    /*
     * 这不是库函数，它仅供内核使用。因此我们不关心小于 1970 年的年份等，但假定一切均很正常。
     * 同样，时间区域 TZ 问题也先忽略。我们只是尽可能简单地处理问题。最好能找到一些公开的库函数
     * （尽管我认为 minix 的时间函数是公开的）。
     * 另外，我恨那个设置 1970 年开始的人 - 难道他们就不能选择从一个闰年开始？我恨格里高利历、
     * 罗马教皇、主教，我什么都不在乎。我是个脾气暴躁的人。
     */
20 #define MINUTE 60                    // 1 分钟的秒数。
21 #define HOUR (60*MINUTE)             // 1 小时的秒数。
22 #define DAY (24*HOUR)                // 1 天的秒数。
23 #define YEAR (365*DAY)               // 1 年的秒数。
24
25 /* interestingly, we assume leap-years */
   /* 有趣的是我们考虑进了闰年 */
   // 下面以年为界限，定义了每个月开始时的秒数时间。
26 static int month[12] = {
27         0,
28         DAY*(31),
29         DAY*(31+29),
30         DAY*(31+29+31),
31         DAY*(31+29+31+30),
32         DAY*(31+29+31+30+31),
33         DAY*(31+29+31+30+31+30),
34         DAY*(31+29+31+30+31+30+31),
35         DAY*(31+29+31+30+31+30+31+31),
36         DAY*(31+29+31+30+31+30+31+31+30),
37         DAY*(31+29+31+30+31+30+31+31+30+31),
38         DAY*(31+29+31+30+31+30+31+31+30+31+30)
39 };
40
   // 该函数计算从 1970 年 1 月 1 日 0 时起到开机当日经过的秒数，作为开机时间。
   // 参数 tm 中各字段已经在 init/main.c 中被赋值，信息取自 CMOS。
41 long kernel_mktime(struct tm * tm)
```

```
42 {
43         long res;
44         int year;
45
   // 首先计算 70 年到现在经过的年数。因为是 2 位表示方式，所以会有 2000 年问题。我们可以
   // 简单地在最前面添加一条语句来解决这个问题：if (tm->tm_year<70) tm->tm_year += 100;
   // 由于 UNIX 计年份 y 是从 1970 年算起。到 1972 年就是一个闰年，因此过 3 年（71，72，73）
   // 就是第 1 个闰年，这样从 1970 年开始的闰年数计算方法就应该是为 1 + (y - 3)/4，即为
   // (y + 1)/4。res = 这些年经过的秒数时间 + 每个闰年时多 1 天的秒数时间 + 当年到当月时
   // 的秒数。另外，month[]数组中已经在 2 月份的天数中包含进了闰年时的天数，即 2 月份天数
   // 多算了 1 天。因此，若当年不是闰年并且当前月份大于 2 月份的话，我们就要减去这天。因
   // 为从 70 年开始算起，所以当年是闰年的判断方法是 (y + 2) 能被 4 除尽。若不能除尽（有余
   // 数）就不是闰年。
46         year = tm->tm_year - 70;
47 /* magic offsets (y+1) needed to get leapyears right.*/
   /* 为了获得正确的闰年数，这里需要这样一个魔幻值(y+1) */
48         res = YEAR*year + DAY*((year+1)/4);
49         res += month[tm->tm_mon];
50 /* and (y+2) here. If it wasn't a leap-year, we have to adjust */
   /* 以及(y+2)。如果(y+2)不是闰年，那么我们就必须进行调整(减去一天的秒数时间)。*/
51         if (tm->tm_mon>1 && ((year+2)%4))
52                 res -= DAY;
53         res += DAY*(tm->tm_mday-1);          // 再加上本月过去的天数的秒数时间。
54         res += HOUR*tm->tm_hour;             // 再加上当天过去的小时数的秒数时间。
55         res += MINUTE*tm->tm_min;            // 再加上 1 小时内过去的分钟数的秒数时间。
56         res += tm->tm_sec;                   // 再加上 1 分钟内已过的秒数。
57         return res;                          // 即等于从 1970 年以来经过的秒数时间。
58 }
59
```