kernel\_mktime() 详解 —— Linux-0.11 学习笔记（四）

在init/main.c文件中，有一个函数static void time\_init(void)

该函数读取 CMOS 实时时钟信息作为开机时间，并保存到全局变量startup\_time （以秒为单位）中。

static void time\_init(void)

{

struct tm time;

do {

time.tm\_sec = CMOS\_READ(0); //当前时间的秒值，格式均是BCD码

time.tm\_min = CMOS\_READ(2); //当前时间的分钟值

time.tm\_hour = CMOS\_READ(4); //当前时间的小时值

time.tm\_mday = CMOS\_READ(7); //当前的日

time.tm\_mon = CMOS\_READ(8); //当前的月

time.tm\_year = CMOS\_READ(9); //当前的年，只有后2位数，例如97表示1997年

} while (time.tm\_sec != CMOS\_READ(0));

BCD\_TO\_BIN(time.tm\_sec); // 转换成二进制数值

BCD\_TO\_BIN(time.tm\_min);

BCD\_TO\_BIN(time.tm\_hour);

BCD\_TO\_BIN(time.tm\_mday);

BCD\_TO\_BIN(time.tm\_mon);

BCD\_TO\_BIN(time.tm\_year);

time.tm\_mon--; // 减一后月份范围是0~11

startup\_time = kernel\_mktime(&time);

}

6~11行：读出当前时间，注意，格式为BCD码值；

13~18行：把BCD码值转换成二进制；

第19行：time.tm\_mon--;这里把月份值减一，为什么这样做，后文会说明。

第20行：调用函数kernel\_mktime()，计算从 1970 年 1 月 1 日 0 时起到此次开机时刻经过的秒数，作为开机时间。

上面的代码就说到这里，CMOS\_READ, BCD\_TO\_BIN等宏定义以后再说。本文想说说kernel\_mktime这个函数。此函数在文件kernel/mktime.c 的第 41 行。kernel/mktime.c这个文件很短，仅有58行。

1. 宏定义

#define MINUTE 60 //1分钟经过的秒数

#define HOUR (60\*MINUTE) //1小时经过的秒数

#define DAY (24\*HOUR) //一天经过的秒数

#define YEAR (365\*DAY) //一年经过的秒数（不考虑闰年）

2. 从1.1到x.1经过的秒数

static int month[12] = {

0, //[0] 1.1-1.1

DAY\*(31), //[1] 1.1-2.1

DAY\*(31+29), //[2] 1.1-3.1

DAY\*(31+29+31), //[3] 1.1-4.1

DAY\*(31+29+31+30), //[4] 1.1-5.1

DAY\*(31+29+31+30+31), //[5] 1.1-6.1

DAY\*(31+29+31+30+31+30), //[6] 1.1-7.1

DAY\*(31+29+31+30+31+30+31), //[7] 1.1-8.1

DAY\*(31+29+31+30+31+30+31+31), //[8] 1.1-9.1

DAY\*(31+29+31+30+31+30+31+31+30), //[9] 1.1-10.1

DAY\*(31+29+31+30+31+30+31+31+30+31), //[10] 1.1-11.1

DAY\*(31+29+31+30+31+30+31+31+30+31+30) //[11] 1.1-12.1

};

假如当前是4月，问：从本年1月1日起到4月1日，经过了多少秒？

可以先算出经过了多少天，再把天数乘以DAY（见宏定义）。如果用 D(m) 表示月份m的总天数，那么答案就是：

( D(1) + D(2) + D(3) ) \* DAY

把上面的问题一般化为：假如当前是x月，问：从本年1月1日起到x月1日，经过了多少秒？

答案是：

( D(1) + D(2) + D(3) + ... + D(x-1) ) \* DAY

思路就是这样， Linus 用的是查表法，于是就有了上面的数组。比如从CMOS中读出的是8月份，那么答案就是month[7]；再比如读出的是12月份，那么答案就是month[11]；再来个特殊情况，比如读出的是1月份，那么就是0，即month[0]. 看出来了吧，索引值比真实的月份值少1，这就是time.tm\_mon--;的原因。

注意，代码中假设今年是闰年，即2月份有29天。

3. 结构体struct tm

struct tm {

int tm\_sec;

int tm\_min;

int tm\_hour;

int tm\_mday;

int tm\_mon;

int tm\_year; //以上6行不用多说，用来保存读出来的年月日时分秒

int tm\_wday;

int tm\_yday;

int tm\_isdst; //夏令时标志

};

8~10行：这3个成员好像没有用到。

4. kernel\_mktime()函数

long kernel\_mktime(struct tm \* tm)

{

long res;

int year;

year = tm->tm\_year - 70; //计算70年到现在（今年的1.1）经过的年数

/\* magic offsets (y+1) needed to get leapyears right.\*/

res = YEAR\*year + DAY\*((year+1)/4); //把年换算成秒，把闰年多出来的天也换算成秒

res += month[tm->tm\_mon]; //把今年的1.1到现在的x.1换算成秒

/\* and (y+2) here. If it wasn't a leap-year, we have to adjust \*/

if (tm->tm\_mon>1 && ((year+2)%4))

res -= DAY;

res += DAY\*(tm->tm\_mday-1); //不算今天

res += HOUR\*tm->tm\_hour;

res += MINUTE\*tm->tm\_min;

res += tm->tm\_sec;

return res;

}

总的来说，计算的方法是先整后零：从1970.1.1算到今年的1.1，再算到本月1日，再算到今天的0点，再到此刻的时分秒。

第6行：因为是从1970年算起，且tm->tm\_year中是年份的末2位，所以要减去70。举例来说，如果是1998年，那么tm->tm\_year = 98，year = 28.

注意：因为年份是 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（比如第5行）添加一条语句来解决这个问题：

if(tm->tm\_year < 70)

tm->tm\_year += 100;

1

2

推导过程：

20xx−1970=(2000+xx)−(1900+70)=2000+xx−1900−70=100+xx−70

举例来说，假如是2007年，那么tm->tm\_year = 7，执行上面的2行语句后，tm->tm\_year = 107，再执行原来的第6行，year = 37；

第8行：res = YEAR\*year + DAY\*((year+1)/4);

(year+1)/4表示从1970年1.1到今年的1.1，经过了几个闰年。注意：1972年是闰年。

为什么是这个式子，或者说为什么它是对的，列出来找找规律就明白了。

读出的年份 year的值 经过的闰年数 备注

1970,1971,1972 0,1,2 0 因为截至今年的1.1，所以即使读出1972年，也不能算是经过了闰年，后面的1976、1980等同理

1973,1974,1975,1976 3,4,5,6 1 如果读出1973~1976，因为经过了1972，所以算为1

1977,1978,1979,1980 7,8,9,10 2 如果读出1977~1980，因为经过了1972和1976，所以算为2

通过上表的中间2列，可以归纳出公式：

经过的闰年数=(year+1)/4

第9行：res += month[tm->tm\_mon];在前文第2节已经解释了。

到目前为止（代码第10行之前），已经计算了1970年1月1日0时到今年本月1日0时经历的秒数。

11~12行：

if (tm->tm\_mon>1 && ((year+2)%4))

res -= DAY;

1

2

如果此表达式(year+2)%4)取值为0，则说明是闰年（观察上表中带下划线的数字就可以得出）；取值不为0，说明不是闰年；

如果tm->tm\_mon>1成立，说明现在的月份是3~12（注意之前的减一）；否则现在的月份是1或者2；

以上2个条件，组合起来有4种情况。

现在的月份 今年是闰年吗？ 结论

1,2 否 因为算到本月1日，所以不牵扯2.29；

1,2 是 同上

3-12 否 多算了2.29，所以要减去1天

3-12 是 是闰年，算2.29没有错

根据上面的分析，只有表格第3行这种情况需要减去1天，于是就有了上面的代码。

剩下的代码就很好理解了，这里不再赘述。

---------------------

作者：ARM的程序员敲着诗歌的梦

来源：CSDN

原文：https://blog.csdn.net/longintchar/article/details/79596747

版权声明：本文为博主原创文章，转载请附上博文链接！