第10章 移植U-Boot

## 10.1 RTC的作用及时间表示

“RTC”的英文全称是Real-Time Clock，翻译过来是实时时钟芯片。实时时钟芯片是日常生活中应用最为广泛的电子器件之一，它为人们或者电子系统提供精确的实时时间。实时时钟芯片通过引脚对外提供时间读写接口，通常内部带有电池，保证在外部系统关电时，内部电路正常工作，时间正常运行。不同的时钟芯片内部机制不一样，时间数据存储格式、读写操作方式也不一样，Linux系统和驱动封装了不同时钟芯片的操作细节，为应用程序提供了统一的时间操作接口。

那么在Linux世界里，时间是怎么表示的呢？是不是与人们一样用年月日+时分秒来表示时间呢？聪明的程序员自然不会让计算机这么做，正所谓越简单越科学，直接用一个整数表示时间，这个整数代表当前与Epoch Time的时间差(以秒为单位)。Epoch Time 是指一个特定的时间：1970年1月1日0时0分0秒。假设现在距离1970年1月1日0时0分0秒走了N秒，在Linuxx系统里，时间数值就是N。

那么问题来了，为什么要从1970年1月1日0时0分0秒开始呢?那年发生了什么，以至于Unix系统以它作“纪元”。原来Unix就是在那个时代产生的，1969年发布的雏形，最早是基于硬件60Hz的时间计数。1971年底出版的《Unix Programmer's Manual》里定义的Unix Time是以1971年1月1日00:00:00作为起始时间，每秒增长60。后来考虑到如果每秒60个数字，则1.1年后时间值就到达最大，于是改成以秒为计数单位，时间能表示到68.1年之长，就不在乎起始时间是1970还是1971年，遂改成人工记忆、计算比较方便的1970年。于是Unix的世界开启了“纪元”，Unix时间戳也就成为了一个专有名称。后Linux系统沿用了这种定义时间的方式。

当时计算机操作系统是32位，时间也是用一个32位的有符号数来表示，数据取值范围为-2147483648~2147483647，也就是说时间最大值只能取到2147483647（秒），换算成年也即2147483647÷365÷24÷60÷60=68.1年，也就是说32位能表示的最长时间是1970+68.1=2038年。精确点讲，2038年01月19日03时14分07秒，时间便会达到最大值，为0x7FFFFFFF。过了这个时间点，下一秒时间值便会变为0x80000000，算下来也就是1901年12月13日20时45分52秒，这样便会出现时间回归的现象，很多系统便会运行异常。

上边说过了，那是Unix系统“元年”时候的事情，32位的时间已经足以解决当时的问题了。现在主流CPU都是64位的，使用64位的数据表示时间也是顺其自然的事，用64位的有符号数来表示时间，可以表示到292，277，026，596年12月4日15时30分08秒，相信我们再也不用愁时间回归的问题了。

## 10.2 RTC的操作命令

弄清楚了时间怎么表示之后，Linux是怎么使用和维护时间的呢？我们怎么通过Linux操作时间呢？

### 10.2.1系统时间和硬件时间

在Linux中有系统时钟与硬件时钟两种时钟。系统时间是由CPU主芯片的定时器进行维护的时间，一般情况下都会选择芯片上精度最高的定时器作为系统时间的定时基准，以避免在系统运行较长时间后出现大的时间偏移。特点是系统掉电后，系统时间将丢失。硬件时钟是指系统中包含的RTC芯片内部所维护的时间。RTC芯片都有电池+系统电源的双重供电机制，在系统正常工作时由系统供电，在系统掉电后由电池进行供电。因此系统电源掉电后RTC时间仍然能够正常运行，Linux系统中硬件时钟的基本目的是在Linux不运行时保持时间。

在Linux启动时，将系统时间从硬件时钟初始化，然后不再使用硬件时钟。在系统开机时，由Linux操作系统从RTC芯片读取硬件时间后，由CPU内部定时器维护时间运行。此后操作系统使用的时间都是系统时间，如果没有显式的通过命令去控制RTC的读写操作，系统将不会再从RTC中去获取或者同步设置时间。

### 10.2.2系统时间操作命令

查看系统时间：

date

Sat May 1 08:11:19 EDT 2020

格式化输出：

date +"%Y-%m-%d"

2020-05-01

2秒后输出：

date -d "2 second" +"%Y-%m-%d %H:%M.%S"

2020-05-01 14:21.31

显示1234567890秒的时间：

date -d "1970-01-01 1234567890 seconds" +"%Y-%m-%d %H:%m:%S"

2009-02-13 23:02:30

普通转格式：

date -d "2009-05-01" +"%Y/%m/%d %H:%M.%S"

2020/05/01 00:00.00

输出其他日期：

date -d "+1 day" +%Y%m%d #显示后一天的日期

date -d "-1 day" +%Y%m%d #显示前一天的日期

date -d "-1 month" +%Y%m%d #显示上一月的日期

date -d "+1 month" +%Y%m%d #显示下一月的日期

date -d "-1 year" +%Y%m%d #显示前一年的日期

date -d "+1 year" +%Y%m%d #显示下一年的日期

设置系统时间：

date -s 20200501 #设置成20200501，这样会把具体时间设置成空00:00:00

date -s 01:01:01 #设置具体时间，不会对日期做更改

date -s "01:01:01 2020-05-01" #这样可以设置全部时间

date -s "01:01:01 20200501" #这样可以设置全部时间

date -s "2020-05-01 01:01:01" #这样可以设置全部时间

date -s "20200501 01:01:01" #这样可以设置全部时间

命令更多参数使用方法可访问：https://www.cnblogs.com/machangwei-8/p/10352546.html

### 10.2.3硬件时间操作命令

显示硬件时间：

hwclock或 hwclock -r 或 hwclock --show

2000年04月11日 星期二 13时24分35秒 -0.109687 seconds

设置硬件时钟：

hwclock --set --date '2015-04-11 13:36:11'11

将系统时钟同步到硬件时钟：

hwclock -w

将硬件时钟同步到系统时钟：

hwclock -s

命令更多参数使用方法可访问：https://www.cnblogs.com/wj78080458/p/9806774.html

## 10.3 RTC的数据结构和函数

在Linux环境中，我们学会了使用命令，修改系统时间和硬件时间。在编程时我们当然可以直接使用system系统调用来操作时间，但是这样既显得不够专业，也不能满足大部分需求，因为很多情况下我们不只是要修改时间，而是要对时间进行运算处理。

RTC编程，重点是学习时间相关的结构体和相关操作函数。

### 10.3.1时间相关的数据结构

在C语言涉及中经常需要定时触发事件，涉及到获取系统时间，其结构体类型有多种。Linux系统下,与时间有关的数据类型定义在头文件 /usr/include/sys/time.h 中：

只要有以下几种时间相关的数据类型：

#### 1.time\_t 类型：长整型

一般用来表示从Epoch Time（1970年1月1日午夜(00:00:00)）以来的秒数，单位为秒。

#define \_TIME\_T

typedef long time\_t;          
 #endif

由函数time\_t time(time\_t\* lpt)来获取time\_t 数据，函数返回自Epoch Time（1970年1月1日午夜(00:00:00)）起经过的时间，以秒为单位。如果 lpt不为空，则返回值也存储在lpt指向的变量中。

示例：

time\_t t = time(NULL);

#### 2.struct timeb 结构

它有四个成员，一个是秒，另一个是毫秒。

struct timeb{

time\_t  time;

unsigned short millitm;

short   timezone;

short   dstflag;                  
  };

time是从Epoch Time（1970年1月1日午夜(00:00:00)）起累计的秒数。

millitm是一秒内的毫秒数。

dstflag不为0，说明这是夏令时时间。

timezone是UTC时间和本地时间的相差分钟数。

由函数int ftime(struct timeb \*tp) 来获取timeb，调用成功返回0，调用失败返回-1。

示例：

struct timeb tp;

ftime(&tp);

#### 3.struct  timeval 和struct timezone结构

timeval 有两个成员，一个是秒，另一个表示微秒。

  struct  timeval{

long  tv\_sec；    /\*秒\*/

long  tv\_usec；/\*微秒\*/

 };

tv\_sec为Epoch Time到创建struct timeval时的秒数，tv\_usec为微秒数，即秒后面的零头。比如当tv\_sec为1234567890，tv\_usec为1234，即当前时间距Epoch时间1234567890秒，1234微秒。

struct timezone{

int tz\_minuteswest;/\*和greenwich 时间差了多少分钟\*/

int tz\_dsttime; /\*type of DST correction\*/

};  
tz\_minuteswest表示当前系统所在时区和UTC的时间差，tz\_minuteswest以分钟计算。比如北京GMT+8区，tz\_minuteswest为-480。tz\_dsttime的定义为日光节约时间（DST，也就是夏令时。

由函数int gettimeofday(struct timeval\*tv,struct timezone \*tz )来获取timeval和timezone，在gettimeofday()函数中tv或者tz都可以为空。如果为空则就不返回其对应的结构体。函数执行成功后返回0，失败后返回-1，错误代码存于errno中。

示例：

struct timeval tv;

gettimeofday(&tv, NULL);

#### 4.struct  tm 结构

 struct tm {

　　int tm\_sec; /\* 秒–取值区间为[0,59] \*/

　　int tm\_min; /\* 分 - 取值区间为[0,59] \*/

　　int tm\_hour; /\* 时 - 取值区间为[0,23] \*/

　　int tm\_mday; /\* 一个月中的日期 - 取值区间为[1,31] \*/

　　int tm\_mon; /\* 月份（从一月开始，0代表一月） - 取值区间为[0,11] \*/

　　int tm\_year; /\* 年份，其值从1900开始 \*/

　　int tm\_wday; /\* 星期–取值区间为[0,6]，其中0代表星期天，1代表星期一，以此类推 \*/

　　int tm\_yday; /\* 从每年的1月1日开始的天数–取值区间为[0,365]，其中0代表1月1日，1代表1月2 日，以此类推 \*/

　　int tm\_isdst; /\* 夏令时标识符，实行夏令时的时候，tm\_isdst为正。不实行夏令时的进候，tm\_isdst为0； 不了解情况时，tm\_isdst()为负。\*/

};

int   tm\_sec 代表目前秒数，正常范围为0-59

int   tm\_min 代表目前分数，范围0-59

int   tm\_hour 从午夜算起的时数，范围为0-23

int   tm\_mday 目前月份的日数，范围01-31

int   tm\_mon 代表目前月份，从一月算起，范围从0-11

int   tm\_year 从1900 年算起至今的年数

int   tm\_wday 一星期的日数，从星期一算起，范围为0-6

int   tm\_yday 从今年1月1日算起至今的天数，范围为0-365

int   tm\_isdst 日光节约时间的旗标

由函数struct tm\* gmtime(const time\_t\*timep)解析得到tm，gmtime()将参数timep 所指的time\_t 数据类型中的信息转换成真实世界所使用的时间日期表示方法，然后将结果由结构tm的指针返回。  
    示例：

struct tm\* tm =NULL ;

time\_t t = time(NULL);

tm = gmtime(&t);

### 10.3.2时间相关的函数

#### 1.时间格式化函数

strftime()函数原型：size\_t strftime(char \*str,size\_t max,char \*fmt,struct tm \*tp);函数识别以百分号(%)开始的格式命令集合，其格式由fmt来指定,可以使用strftime 函数将时间格式转化为我们想要的格式（其输出结果为字符串），strftime有点像sprintf。

str 表示返回的时间字符串

count要写入的字节的最大数量

format 格式字符串由零个或多个转换符和普通字符(除%)

tm 输入时间

格式命令，是区分大小写的：

%a 星期几的简写

%A 星期几的全称

%b 月分的简写

%B 月份的全称

%c 标准的日期的时间串

%C 年份的后两位数字

%d 十进制表示的每月的第几天

%D 月/天/年

%e 在两字符域中，十进制表示的每月的第几天

%F 年-月-日

%g 年份的后两位数字，使用基于周的年

%G 年分，使用基于周的年

%h 简写的月份名

%H 24小时制的小时

%I 12小时制的小时

%j 十进制表示的每年的第几天

%m 十进制表示的月份

%M 十时制表示的分钟数

%n 新行符

%p 本地的AM或PM的等价显示

%r 12小时的时间

%R 显示小时和分钟：hh:mm

%S 十进制的秒数

%t 水平制表符

%T 显示时分秒：hh:mm:ss

%u 每周的第几天，星期一为第一天 （值从0到6，星期一为0）

%U 第年的第几周，把星期日做为第一天（值从0到53）

%V 每年的第几周，使用基于周的年

%w 十进制表示的星期几（值从0到6，星期天为0）

%W 每年的第几周，把星期一做为第一天（值从0到53）

%x 标准的日期串

%X 标准的时间串

%y 不带世纪的十进制年份（值从0到99）

%Y 带世纪部分的十进制年份

%z，%Z 时区名称，如果不能得到时区名称则返回空字符。

%% 百分号

示例：

time\_t t = time(NULL);

struct tm \*info;

info = gmtime(&t);

strftime(buffer, 80, "%Y-%m-%d %H:%M:%S", info);

printf("格式化的日期 & 时间 : |%s|\n", buffer );

#### 2.localtime函数

localtime（）函数原型：struct tm \*localtime(const time\_t \*timer) 使用 timer 的值来填充 tm 结构，timer 的值被解析为 tm 结构，并用本地时区表示

示例：

time\_t rawtime;

struct tm \*info;

time( &rawtime );

info = localtime( &rawtime );

#### 3.mktime函数

mktime()函数原型： time\_t mktime(struct tm \*timeptr) 把 timeptr 所指向的结构转换为一个依据本地时区的 time\_t 值。函数将参数timeptr所指的tm结构数据转换成从公元1970年1月1日0时0分0 秒算起至今的UTC时间所经过的秒数。该函数返回一个 time\_t 值，该值对应于以参数传递的日历时间。如果发生错误，则返回 -1 值。

示例：

time\_t rawtime;

struct tm \* timeinfo;

int year, month ,day;

/\* 获取当前时间信息，并修改用户输入的输入信息 \*/

time ( &rawtime );

timeinfo = localtime ( &rawtime );

timeinfo->tm\_year-= 1;//去年的今天

rawtime = mktime ( timeinfo );

#### 4.asctime函数

asctime()函数原型： char \*asctime(const struct tm \*timeptr) ；函数返回一个指向字符串的指针，它代表了结构 struct timeptr 的日期和时间。包含了可读格式的日期和时间信息 Www Mmm dd hh:mm:ss yyyy，其中，Www 表示星期几，Mmm 是以字母表示的月份，dd 表示一月中的第几天，hh:mm:ss 表示时间，yyyy 表示年份。

示例：

struct tm t;

t.tm\_sec = 10;

t.tm\_min = 10;

t.tm\_hour = 6;

t.tm\_mday = 25;

t.tm\_mon = 2;

t.tm\_year = 89;

t.tm\_wday = 6;

puts(asctime(&t));

#### 5.ctime函数

ctime()函数原型：char \*ctime(const time\_t \*timer); 可以把time函数得到的结果转换成一个时间字符串， 返回一个表示当地时间的字符串，当地时间是基于参数 timer。返回的字符串格式如下： Www Mmm dd hh:mm:ss yyyy 其中，Www 表示星期几，Mmm 是以字母表示的月份，dd 表示一月中的第几天，hh:mm:ss 表示时间，yyyy 表示年份。调用ctime(t)等价于asctime(localtime(t))。

示例：

time\_t curtime;

time(&curtime);

printf("当前时间 = %s", ctime(&curtime));

## 10.4 RTC时间的编程实例

嵌入式Linux环境下，RTC时间编程与桌面Linux环境下的编程是一样的。本文在桌面Linux下编程，实现时间显示、计算等功能。编译后的程序名为“mytime”，进入程序后，使用不同命令完成不同功能，主要命令见下表：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 命令 | 功能 | 示例 |
| 1 | p | 在终端显示当前时间 | p |
| 2 | y | 在终端显示昨天日期 | y |
| 3 | n | 在终端显示现在距新年的天数 | n |
| 4 | a | 根据输入的出生年代，在终端显示年纪 | a 2001 |
| 5 | e | 退出程序 | e |

在main函数中，首先打印函数的使用手册，然后循环接收用户输入的命令，根据命令调用相应的函数：

32 //打印使用手册

33 printf( "\n\n"\

34 "Usage:\np\n"\

35 "y\n"\

36 "n\n"\

37 "a 2001\n"\

38 "e\n"\

39 "p:在终端显示当前时间\ny:查看昨天日期\nn:查看现在距新年的天数\n"\

40 "a:根据输入的出生年计算出年纪\ne:退出程序\n"\

41 );

42

43 //主程序中循环接收输入的命令，根据不同命令执行不同函数

44 while (1){

45 if (c !='\n')

46 printf("\n请输入命令：");

47 scanf("%c",&c);

48 switch(c){

49 case 'p'://在终端显示当前时间

50 displaydate();

51 break;

52 case 'y'://显示昨天日期

53 displayyesterday();

54 break;

55 case 'n'://显示现在距新年的天数

56 displaynewyear();

57 break;

58 case 'a'://根据输入的出生年计算出年纪

59 scanf("%d",&age);

60 displayage(age);

61 break;

62 case 'e'://退出程序

63 exit(0);

64 break;

65

66 default : /\* 可选的 \*/

67 break;

68 }

69 }

实现在终端显示当前时间的函数：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

72 \* 函数名称： displaydate

73 \* 功能描述： 在终端打印当前时间信息

74 \* 输入参数： 无

75 \* 输出参数： 无

76 \* 返 回 值： 无

77 \* 2020/05/10 V1.0 yanxni 创建

78 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

79 void displaydate(){

80 struct tm \*ptr;

81 time\_t lt;

82

83 /\*获取日历时间\*/

84 lt = time(NULL);

85

86 /\*转化为本地时间\*/

87 ptr = localtime(&lt);

88

89 /\*以本地时间的字符串方式打印\*/

90 printf("%s\n",ctime(&lt));

91

92 /\*以本地时间的字符串方式打印\*/

93 printf("%s\n",asctime(ptr));

94

95 }

实现在终端显示昨天日期的函数：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

97 \* 函数名称： displayyesterday

98 \* 功能描述： 在终端打印昨天的日期

99 \* 输入参数： 无

100 \* 输出参数： 无

101 \* 返 回 值： 无

102 \* 2020/05/10 V1.0 yanxni 创建

103 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

104 void displayyesterday(void){

105 struct tm \*ptr;

106 time\_t lt;

107

108 /\*获取日历时间\*/

109 lt = time(NULL);

110 lt -= 24\*60\*60;

111

112 /\*转化为本地时间\*/

113 ptr = localtime(&lt);

114

115 /\*以本地时间的字符串方式打印\*/

116 printf("昨天是%d年%d月%d日\n",ptr->tm\_year + 1900,ptr->tm\_mon + 1,ptr->tm\_mday);

117 }

实现在终端显示现在距新年的天数的函数：

118 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

119 \* 函数名称： displaynewyear

120 \* 功能描述： 在终端打印距离新年的天数

121 \* 输入参数： 无

122 \* 输出参数： 无

123 \* 返 回 值： 无

124 \* 2020/05/10 V1.0 yanxni 创建

125 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

126 void displaynewyear(void){

127 struct tm \*ptr;

128 time\_t lt,lt2;

129 int date;

130

131 /\*获取日历时间\*/

132 lt = time(NULL);

133

134 /\*转化为本地时间\*/

135 ptr = localtime(&lt);

136 /\*构造新年的本地时间\*/

137 ptr->tm\_year += 1;

138 ptr->tm\_mon = 0;

139 ptr->tm\_mday =1;

140 ptr->tm\_hour =0;

141 ptr->tm\_min =0;

142 ptr->tm\_sec =0;

143

144 lt2 = mktime(ptr);

145 date = (lt2-lt)/(24\*60\*60);

146

147 printf("距离新年还有%d天\n",date);

148 }

实现在根据输入的出生年代在终端显示年纪的函数：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

151 \* 函数名称： displayage

152 \* 功能描述： 在终端打印年纪

153 \* 输入参数： 出生年代

154 \* 输出参数： 无

155 \* 返 回 值： 无

156 \* 2020/05/10 V1.0 yanxni 创建

157 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

158 void displayage(int year){

159 struct tm \*ptr;

160 time\_t lt;

161

162 /\*获取日历时间\*/

163 lt = time(NULL);

164

165 /\*转化为本地时间\*/

166 ptr = localtime(&lt);

167

168 printf("你的年龄是：%d岁\n",ptr->tm\_year +1900 - year );