<https://blog.csdn.net/rookie_zhu/article/details/8610219>

概述

世界标准时间（Coordinated Universal Time，UTC），也就是大家所熟知的格林威治标准时间（Greenwich Mean Time，GMT）。世界各地时间也世界标准时间为基准划分为不同的时区，例如，中国的北京时间与UTC的时差为+8，也就是UTC+8。美国是UTC-5。

Calendar Time：日历时间，是用“从一个标准时间点到此时的时间经过的秒数”来表示的时间。无论哪一个时区，在同一时刻对同一个标准时间点来说，日历时间都是一样的。日历时间返回自1970-1-1:00:00:00以来所经过的秒数累计值。

**跟日期时间相关的shell命令**

$ date                    // 显示当前日期

$ time                   // 显示程序运行的时间

$ hwclock             // 显示与设定硬件时钟

$ clock                  // 显示与设定硬件时钟，是hwclock的链接文件

$ cal                      // 显示日历

（1）date 显示或设置当前日期时间

$ date             显示当前日期时间 -- 中国北京时间 CST China Standard Time UTC+8:00

2008年 05月 01日 星期四 04:28:27 CST

$ date –u        显示当前日期时间 -- 世界标准时间 UTC

2008年 04月 30日 星期三 20:29:23 UTC

以上两个时间相比有8个小时的时差

$ date –R        显示当前日期时间 – RFC格式

Thu, 01 May 2008 04:30:25 +0800

$ date -s 20080501        设置日期

$ date -s 20:40:30          设置时间

（2）time 显示程序运行时消耗的实际时间，用户CPU时间和系统CPU时间。

$ time a.out                  可执行程序a.out

real    0m10.081s              程序开始运行到结束的时间

user    0m0.000s        用户CPU时间，

sys     0m0.004s        系统CPU时间

用户CPU时间等于times函数返回的struct tms中的tms\_utime 和 tms\_cutime和。

系统CPU时间等于times函数返回的struct tms中的tms\_stime 和 tms\_cstime和。

（3）hwclock       显示与设定硬件时钟

在Linux中有硬件时钟与系统时钟等两种时钟。硬件时钟是指主机板上的时钟设备，也就是通常可在BIOS画面设定的时钟。系统时钟则是指 kernel中的时钟。当Linux启动时系统时钟会去读取硬件时钟的设定，之后系统时钟即独立运作。所有Linux相关指令与函数都是读取系统时钟的设定。

# hwclock –show          显示硬件时钟的时间与日期

# hwclock –hctosys             将硬件时钟调整为与目前的系统时钟一致

# hwclock –systohc             将硬件时钟调整为与目前的系统时钟一致

# hwclock --set --date="20080430 21:30:30"              设定硬件时钟

# hwclock                    同hwclock –show

Clock 命名是hwclock的链接文件

$ ls -al /sbin/clock

lrwxrwxrwx 1 root root 7 03-08 23:59 /sbin/clock -> hwclock

（4）显示日历

$ cal                            显示本年本月的日历

$ cal month year    显示指定年月的日历： cal 4 2008

1.2 跟日期时间有关的数据结构

1.2.1 **clock\_t**结构

程序开始运行到此时所经过的CPU时钟计时单元数用clock数据类型表示。

typedef long clock\_t;

#define CLOCKS\_PER\_SEC ((clock\_t)1000)      // 每个时钟单元是1毫秒

1.2.2 time\_t 结构

日历时间（Calendar Time）是通过time\_t数据类型来表示的，用time\_t表示的时间（日历时间）是从一个时间点（1970年1月1日0时0分0秒）到此时的秒数。

typedef long time\_t;                     // 时间值

1.2.3 tm结构

通过tm结构来获得日期和时间

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，其值等于实际年份减去1900 \*/

        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 \*/

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

        int tm\_isdst;   /\* 夏令时标识符，夏令时tm\_isdst为正；不实行夏令时tm\_isdst为0；\*/

};

1.2.4 tms结构

保存着一个进程及其子进程使用的CPU时间

struct tms{

       clock\_t tms\_utime;

       clock\_t tms\_stime;

       clock\_t tms\_cutime;

       clock\_t tms\_cstime;

}

1.2.5 Utimbuf结构

struct utimbuf{

       time\_t     actime;           // 存取时间

       time\_t     modtime;        // 修改时间

}

**文件的时间**

st\_atime         文件数据的最后存取时间

st\_mtime        文件数据的最后修改时间

st\_ctime         文件数据的创建时间

1.2.5 timeval结构

struct timeval  
{  
    time\_t tv\_sec;  
    susecond\_t tv\_usec; //当前妙内的微妙数  
};

**1.2.6 timer\_struct**结构

struct timer\_struct {

    unsigned long expires; //定时器被激活的时刻

    void (\*fn)(void); //定时器激活后的处理函数 }

1.3 跟日期时间相关的函数

1.3.1 clock函数

#include <time.h>

clock\_t clock(void);

返回从程序开始运行到程序中调用clock()函数之间的CPU时钟计时单元数

**例 1：clock函数的例子**

$ vi clock.c

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main(void)

{

    long    loop = 10000000L;

    double duration;

    clock\_t start, end;

    printf("Time to do %ld empty loops is ", loop);

    start = clock();

    while(loop--)   ;

    end = clock();

   duration = (double)(end-start)/CLOCKS\_PER\_SEC;

    printf("%f seconds\n", duration);

    return(0);

}

编译、运行：

$ gcc clock.c -o clock

$ ./clock

Time to do 10000000 empty loops is 0.220000 seconds

1.3.2 time函数

日历时间

#include <time.h>

time\_t time(time\_t \*calptr)；

返回自1970-1-1:00:00:00以来经过的秒数累计值

**例 2：time函数的例子**

#include <stdio.h>

#include <time.h>

int main(void)

{

    time\_t now;

    time(&now);

    printf("now time is %d\n", now);

    return(0);

}

编译、运行：

$ gcc time.c -o time

$ ./time

now time is 1193688148

1.3.3 times函数

程序运行的时间

#include <sys/times.h>

clock\_t times(struct tms \*buf);

返回自系统自举后经过的时钟滴答数

**例 3：times函数的例子**

#include <stdio.h>

#include <sys/times.h>

int main(void)

{

    int i;

    clock\_t start, end;

    struct tms tms\_start, tms\_end;

    start = times(&tms\_start);

    end = times(&tms\_end);

    printf("start clock time : %d\n", start);

    printf("end   clock time : %d\n", end);

    return(0);

}

编译、运行：

$ gcc times.c -o times

$ ./times

Start clock time : 1720654909

End clock time : 1720654909

1.3.4 localtime函数

将日历时间变换成本地时间，考虑到本地时区和夏令时标志。

#include <time.h>

struct tm \*localtime(const time\_t \* calptr);

**例 4：localtime函数的例子**

#include <stdio.h>

#include <time.h>

int main(void)

{

    time\_t now;

    struct tm \*tm\_now;

    time(&now);

    tm\_now = localtime(&now);

    printf("now datetime: %d-%d-%d %d:%d:%d\n", tm\_now->tm\_year, tm\_now->tm\_mon, tm\_now->tm\_mday, tm\_now->tm\_hour, tm\_now->tm\_min, tm\_now->tm\_sec);

    return(0);

}

编译、运行：

$ gcc localtime.c -o localtime

$ ./localtime

now datetime: 107-9-30 5:11:43

1.3.5 gmtime函数

将日历时间变换成国际标准时间的年月日分秒

#include <time.h>

struct tm \*gmtime(const time\_t \*calptr);

**例 5：gmtime函数的例子**

#include <stdio.h>

#include <time.h>

int main(void)

{

    time\_t now;

    struct tm \*tm\_now;

    time(&now);

    tm\_now = gmtime(&now);

    printf("now datetime: %d-%d-%d %d:%d:%d\n", tm\_now->tm\_year, tm\_now->tm\_mon, tm\_now->tm\_mday, tm\_now->tm\_hour, tm\_now->tm\_min, tm\_now->tm\_sec);

    return(0);

}

编译、运行：

$ gcc gmtime.c -o gmtime

$ ./gmtime

now datetime: 107-9-29 21:15:26

1.3.6 mktime函数

以本地时间的年月日为参数，将其变换成time\_t值

#include <time.h>

time\_t mktime(struct tm \*tmptr);

**例 6：mktime函数的例子**

#include <stdio.h>

#include <time.h>

int main(void)

{

    time\_t now, new\_time;

    struct tm \*tm\_now;

    time(&now);

    printf("now time is %ld\n", now);

    tm\_now = localtime(&now);

    new\_time = mktime(tm\_now);

    printf("new time is %ld\n", new\_time);

    return(0);

}

编译、运行：

$ gcc mktime.c -o mktime

$ ./mktime

now time is 1193692806

new time is 1193692806

1.3.7 asctime函数

产生形式的26字节字符串，参数指向年月日等字符串的指针。与date命令输出形式类似

#include <time.h>

char \*asctime(const struct tm \*tmptr);

**例 7：astime函数的例子**

#include <stdio.h>

#include <time.h>

int main(void)

{

    time\_t now;

    struct tm \*tm\_now;

    char \*datetime;

    time(&now);

    tm\_now = localtime(&now);

    datetime = asctime(tm\_now);

    printf("now datetime: %s\n", datetime);

    return(0);

}

编译、运行：

$ gcc asctime.c -o asctime

$ ./asctime

now datetime: Tue Oct 30 05:22:21 2007

1.3.8 ctime函数

产生形式的26字节字符串，参数指向日历时间的指针。

#include <time.h>

char \*ctime(const time\_t \*calptr);

**例 8：ctime函数的例子**

#include <stdio.h>

#include <time.h>

int main(void)

{

    time\_t now;

    char \*datetime;

    time(&now);

    datetime = ctime(&now);

    printf("now datetime: %s\n", datetime);

    return(0);

}

编译、运行：

$ gcc ctime.c -o ctime

$ ./ctime

now datetime: Tue Oct 30 05:23:45 2007

1.3.9 strftime函数

格式化时间输出

#include <time.h>

size\_t strftime(char \*buf,size\_t maxsize,const char \*format,const struct tm \*tmptr);

%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）

%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

**例 9：strftime函数的例子**

#include <stdio.h>

#include <time.h>

int main(void)

{

    time\_t now;

    struct tm \*tm\_now;

    char    datetime[200];

    time(&now);

    tm\_now = localtime(&now);

    strftime(datetime, 200, "%x %X %n%Y-%m-%d %H:%M:%S %nzone: %Z\n", tm\_now);

    printf("now datetime : %s\n", datetime);

    return(0);

}

编译、运行：

$ gcc strftime.c -o strftime

]$ ./strftime

now datetime : 10/30/07 05:41:47

2007-10-30 05:41:47

zone: CST

1.3.10 utime函数

更改文件的存取和修改时间

#include <time.h>

int utime(const char pathname, const struct utimbuf \*times)      ；

返回值：成功返回0，失败返回-1

times 为空指针，存取和修改时间设置为当前时间

**例 10：utime函数的例子**

#include <stdio.h>

#include <time.h>

int main(int argc, char \*argv[])

{

       if(argc < 2){

              fprintf(stderr, "Error: usging command file\_path");

              exit(1);

       }

       utime(argv[1], NULL);

       return(0);

}

编译、运行：

$ touch file1

$ ls -al file1            // 先创建一个文件file1，查看一下他的创建时间

-rw-r--r-- 1 hongdy hongdy 3431 05-01 05:59 file1

$ gcc utime.c –o utime

$ ./utime file1

$ ls -al file1

-rw-r--r-- 1 hongdy hongdy 3431 05-01 06:00 file1

1.3.11 **gettimeofday函数**

取得目前的时间

#include <time.h>

int gettimeofday ( struct& nbsptimeval \* tv , struct timezone \* tz )；

函数说明  gettimeofday()会把目前的时间有tv所指的结构返回，当地时区的信息则放到tz所指的结构中。  
timeval结构定义为:  
struct timeval{  
long tv\_sec; /\*秒\*/  
long tv\_usec; /\*微秒\*/  
};  
timezone 结构定义为:  
struct timezone{  
int tz\_minuteswest; /\*和Greenwich 时间差了多少分钟\*/  
int tz\_dsttime; /\*日光节约时间的状态\*/  
};  
上述两个结构都定义在/usr/include/sys/time.h。tz\_dsttime 所代表的状态如下  
DST\_NONE /\*不使用\*/  
DST\_USA /\*美国\*/  
DST\_AUST /\*澳洲\*/  
DST\_WET /\*西欧\*/  
DST\_MET /\*中欧\*/  
DST\_EET /\*东欧\*/  
DST\_CAN /\*加拿大\*/  
DST\_GB /\*大不列颠\*/  
DST\_RUM /\*罗马尼亚\*/  
DST\_TUR /\*土耳其\*/  
DST\_AUSTALT /\*澳洲（1986年以后）\*/  
   
返回值  成功则返回0，失败返回－1，错误代码存于errno。附加说明EFAULT指针tv和tz所指的内存空间超出存取权限。

times 为空指针，存取和修改时间设置为当前时间

**例 10：gettimeofday函数的例子**

#include<sys/time.h>  
#include<unistd.h>  
main(){  
struct timeval tv;  
struct timezone tz;  
gettimeofday (&tv , &tz);  
printf(“tv\_sec; %d\n”, tv,.tv\_sec)   
printf(“tv\_usec; %d\n”,tv.tv\_usec);  
printf(“tz\_minuteswest; %d\n”, tz.tz\_minuteswest);  
printf(“tz\_dsttime, %d\n”,tz.tz\_dsttime);  
}

编译、运行：

tv\_usec:136996  
tz\_minuteswest:-540  
tz\_dsttime:0

tv\_sec: 974857339

**1.3.12 settimeofday函数**

设置目前时间

#include<sys/time.h>  
#include<unistd.h>  
   
int settimeofday ( const& nbspstruct timeval \*tv,const struct timezone \*tz);  
   
函数说明  settimeofday()会把目前时间设成由tv所指的结构信息，当地时区信息则设成tz所指的结构。详细的说明请参考gettimeofday()。注意，只有root权限才能使用此函数修改时间。  
   
返回值  成功则返回0，失败返回－1，错误代码存于errno。  
   
错误代码  EPERM 并非由root权限调用settimeofday（），权限不够。  
EINVAL 时区或某个数据是不正确的，无法正确设置时间。

times 为空指针，存取和修改时间设置为当前时间

**例 12：settimeofday函数的例子**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
设置操作系统时间  
参数:\*dt数据格式为"2006-4-20 20:30:30"  
调用方法:  
char \*pt="2006-4-20 20:30:30";  
SetSystemTime(pt);  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
int SetSystemTime(char \*dt)  
{  
struct rtc\_time tm;  
struct tm \_tm;  
struct timeval tv;  
time\_t timep;  
sscanf(dt, "%d-%d-%d %d:%d:%d", &tm.tm\_year,  
&tm.tm\_mon, &tm.tm\_mday,&tm.tm\_hour,  
&tm.tm\_min, &tm.tm\_sec);  
\_tm.tm\_sec = tm.tm\_sec;  
\_tm.tm\_min = tm.tm\_min;  
\_tm.tm\_hour = tm.tm\_hour;  
\_tm.tm\_mday = tm.tm\_mday;  
\_tm.tm\_mon = tm.tm\_mon - 1;  
\_tm.tm\_year = tm.tm\_year - 1900;  
  
timep = mktime(&\_tm);  
tv.tv\_sec = timep;  
tv.tv\_usec = 0;  
if(settimeofday (&tv, (struct timezone \*) 0) < 0)  
{  
printf("Set system datatime error!\n");  
return -1;  
}  
return 0;  
}