重庆理工大学/电气学院
CHONGQING UNIVERSITY OF TECHNOLOGY

嵌入式Linux系统开发教程

—基于ARM处理器通用平台（arm9-arm11-contexA系列）

丛超

2025年5月

1 目录操作

2 文件操作

3 设备控制

4 Linux时间编程

01

目录操作

5.1文件和目录

首先直观的感受一下，在终端下输入命令ls -l，部分显示如下所示。

```
root@unbuntu-virtual-machine:/# ls -l
总用量 1943004
drwxr-xr-x   2 root root       4096 3月  10 22:37 bin
drwxr-xr-x   3 root root       4096 3月  10 22:38 boot
drwxrwxr-x   2 root root       4096 3月  10 22:22 cdrom
drwxr-xr-x  18 root root       4100 3月  20 10:57 dev
drwxr-xr-x 124 root root      12288 3月  20 13:41 etc
drwxr-xr-x   5 root root       4096 3月  20 13:23 home
drwxr-xr-x  21 root root       4096 3月  10 22:37 lib
drwxr-xr-x   2 root root       4096 3月  20 13:41 lib64
drwx------   2 root root      16384 3月  10 22:21 lost+found
drwxr-xr-x   3 root root       4096 3月  20 10:47 media
drwxr-xr-x   3 root root       4096 3月  20 10:57 mnt
```

5.1文件和目录

1.drwxr-xr-x：代表的是文件类型和文件权限。常用的文件类型有：

（1）-普通文件，存各种数据。

（2）d目录文件，存结构体，结构体内部标识这个目录中文件名称等信息。

（3）l链接文件，注意：软链接才是文件，而硬链接仅仅是一节点。

（4）c 字符设备，除块设备都是字符设备，没有扇区的概念。

（5）b 块设备，所有存储类的驱动，都称为块设备 ，包含扇区处理。

（6）p 管道设备，是用内核内存模拟的通道。

可以从上述说明看出来，例子中的文件是一个目录文件，原因是第一个符号代表文件类型，d代表此文件是一个目录文件。

5.1文件和目录

2.文件权限：

（1）r为读，二进制权重为100即4。指用户对文件或目录具有查看权限。若用户对某文件或目录不具有读权限，则不能查看其内容。

（2）w为写，二进制权重为010即2。指用户对文件或目录具有写权限。若用户对某文件或目录没有写权限，则不能修改它。

（3）x为执行，二进制权重为001即1。指用户对文件具有执行权限或对目录具有进入权限。若用户对某文件没有执行权限，则不能执行它；若用户对某目录没有执行权限，则不能进入它。

（4）-无操作，二进制权重为0。指用户对文件或目录不赋予某种权限。-wx表示不能读，但可写可执行。

（5）rwx顺序不可改，表示可读可写可执行。

5.1文件和目录

如果一个文件有全部的权限，那么对应8进制里的数是7（4+2+1）。同时读者会发现有多组的rwx，它所表达的不仅仅是它自身的权限。这里涉及到一个分组的概念，分别是创建者（user，u）、同组人（group，g）和其他人（other、o）。

（1）u组：创建者（user）。

（2）g组：创建者所在组的成员（group）。

（3）o组：其它人所具备的权限（other）。

5.1文件和目录

接着文件类型和权限之后是数字2，这个2表示的是文件节点数，此文件是一个目录文件。所以目录的节点数代表着该目录下的文件个数，那么在这里应该是有两个文件。如果此文件不是目录，只是普通文件，那么这个数字就代表硬链接的个数。

硬链接与软连接区别：

（1）链接分为硬链接和软链接(符号链接，即快捷方式)。

（2）硬链接，只是增加一个引用计数，本质上并没有物理上的增加文件。硬链接不是文件。

（3）符号链接，是在磁盘上产生一个文件，这个文件内部写入了一个指向被链接的文件指针。

（4）采用ln指令，用来在文件之间创建链接，默认为创建硬链接（目录不能创建硬链接），使用选项-s创建符号链接。硬链接指向文件本身，符号链接指向文件名称。

5.2目录操作

1.创建目录

头文件

```
#include <sys/stat.h>
```

定义函数

```
int mkdir(const char *path，mode_t mode);
```

函数说明

path:目录名

mode: 模式，即访问权限，包含如下选项

- S_IRUSR　　属主读权限
- S_IWUSR　　属主写权限
- S_IXUSR　　属主执行权限

Example5.2-1：创建目录

```
#include <stdio.h>
#include <sys/stat.h>
#include <errno.h>
#include <string.h>
int main(int argc,char **argv){
    int err=mkdir(argv[1],0666); //0666
    if (err==-1){
        printf("----- mkdir err no=%d,str=%s\n",errno,strerror(errno));
    }
    return 0;
}
```

5.2目录操作

2.删除目录

int rmdir(const char *path);

说明：只能删除空目录。
返回值 成功返回0，失败返回-1。

5.2目录操作

Example5.2-2：获取当前目录

```
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
int main(int argc,char **argv){
        char dir[256];
        getcwd(dir,256);  //设定buf的内存空间为256字节
        printf("---- %s\n",./dir);  //输出当前的工作目录
        return 0;
}
```

5.2目录操作

（2）获取程序运行路径，即应用程序存放的位置

表头文件

#include <unistd.h>

定义函数

int readlink(const char *path,char *buf, size_t bufsiz);

函数说明

path：符号链接，在linux内执行程序都用"/proc/self/exe"符号连接。

buf：用来写入正在执行的文件名（包含绝对路径）的内存。

bufsiz：即指明buf的大小。

返回值 成功则返回符号连接所指的文件路径字符串，失败返回-1。

5.2目录操作

Example5.2-3：获取当前文件或者目录的路径

```
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
int main(int argc,char **argv){
//获取程序目录，执行时在其它目录执行
        char dir[256]={0};
        int err=readlink("/proc/self/exe",dir,256);  //读取程序执行路径
        if (err==-1) return -1;  //读取失败，返回-1
        int i;
        for(i=strlen(dir)-1;i>=0 && dir[i]!='/';i--);
        dir[i]=0;
 //打开程序所在目录中的文件
        char filename[256];

sprintf(filename,"%s/%s",dir,argv[1]);
        printf("---- %s\n",filename);
        FILE *fp=fopen(filename,"r");
        if (fp==NULL){
           printf("---- %s\n",strerror(errno));
        }
        return 0;
    }
```

5.2目录操作

4. 获取目录或文件的状态

表头文件

#include <sys/types.h>

#include <sys/stat.h>

定义函数

（1）读取指定文件或目录的状态信息

int stat(const char *path, struct stat *buf);

（2）读取已打开的文件状态信息

int fstat(int filedes,struct stat *buf);

函数说明

参数：path：路径或文件名

filedes：已打开文件或目录的句柄

buf：是struct stat结构的指针，其结构格式如下：

```
struct stat{
    unsigned short st_mode; //文件保护模式（即文件类型）
    unsigned short st_nlink; //硬链接引用数
    unsigned short st_uid;  //文件的用户标识
    unsigned short st_gid;  //文件的组标识
    unsigned long  st_size;        //文件大小
    unsigned long  st_atime;       //文件最后的访问时间
    unsigned long  st_atime_nsec;        //文件最后的访问时间的秒数的小数
    unsigned long  st_mtime;   //文件最后的修改时间
    unsigned long  st_mtime_nsec;
    unsigned long  st_ctime;       //文件最后状态的改变时间
    unsigned long  st_ctime_nsec;
    ...
```

在struct stat结构中，由st_mode字段记录了文件类型及访问权限，操作系统提供了一系列的宏来判断文件类型。如下：

S_ISREG(mode) //判断是否为普通文件

S_ISDIR(mode) //判断是否为目录文件

S_ISCHR(mode) //判断是否为字符设备文件

S_ISBLK(mode) //判断是否为块设备文件

S_ISFIFO(mode) //判断是否为管道设备文件

S_ISLNK(mode) //判断是否为符号链接

Example5.2-4：判断文件类型

```
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
int main(int argc,char **argv){
 struct stat st;  //定义一个stat类型的结构体，命名为st
 int err= stat(argv[1],&st);  //读取文件或目录状态信息
 if (err==-1) return -1;  //执行失败，返回-1
 if (S_ISDIR(st.st_mode)) printf("isdir\n");  //访问st_mode字段，判断是否为目录文件
 else if (S_ISREG(st.st_mode)) printf("file\n");  //访问st_mode字段，判断是否为普通文件
}
```

02

文件操作

5.3.1基本概念

1.流：

- Linux以字节为单位操作数据，所有的数据都是0或1的序列，如果需要让人读懂的数据，则以字符的方式显示出来，这就是所谓的文本文件。
- 数据不是存在磁盘上后就永远不动，往往要被读入到内存、传送到外部设备、或搬移到其它位置，所以数据不断的在流动。
- 然而不同设备之间的连接方法差异很大，数据读取和写入的方式也不相同，所以Linux定义了流(stream)，建立了一个统一的接口，无论是数据从内存到外设，还是从内存到文件，都使用同一个数据输入输出接口。

2.文件流和标准流

文件操作有两种方法：原始文件I/O、标准I/O。

（1）标准I/O：是标准C的输入输出库，fopen、fread、fwrite、fclose都是标准C的输入输出函数。标准IO都使用FILE * 流对象指针做为操作文件的唯一识别，所以标准I/O是针对流对象的操作，是带缓存(内存)机制的输入输出。标准I/O又提供了3种不同方式的缓冲：

（2）原始I/O：又称文件I/O，是linux操作系统提供的API，称为系统调用，是针对描述符(即一个编号)操作的，是无缓

3.标准输入，标准输出，标准错误

当Linux执行一个程序的时候，会自动打开三个流，标准输入(standard input)，标准输出(standard output)，标准错误(standard error)。

5.3.2检查文件及确定文件的权限

头文件

```
#include <unistd.h>
```

定义函数

```
int access(const char *pathname, int mode);
```

函数说明

参数：pathname:包含路径的文件名

mode: 模式，有5种模式：

（1）0 检查文件是否存在

（2）1 检查文件是否可执行x

（3）2 检查文件是否可写w

（4）4 检查文件是否可读r

（5）6 检查评议件是否可读写w+r

返回值：0真，非零为假，无论是否为真，这个函数都会向标准错误发送信号，指明执行情况。

5.3.3创建文件

表头文件

#include <fcntl.h>

定义函数

int creat(const char *pathname, mode_t mode);

参数：pathname 包含路径的文件名。

mode为模式，即访问权限，包含如下选项：

(1) S_IRUSR，属主读权限；

(2) S_IWUSR，属主写权限；

(3) S_IXUSR，属主执行权限。

返回值：成功，返回文件描述符，失败时返回-1。

Example5.3.3-1：创建文件

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
int main(int argc,char **argv){
    int fd=-1;      //定义文件描述符fd为-1
    fd=creat(argv[1],00777);  //调用creat函数创建文件
    if (fd<0){
        printf("errno: %s\n",strerror(errno));  //创建失败，返回-1
    }
    else {
        printf("ok : %d\n",fd); //创建成功，返回文件描述符
        close(fd);
    }
    return 0;
}
```

5.3.4打开文件

表头文件

```
#include <fcntl.h>
```

定义函数

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

以上所有选项可以组合使用，例如：

- O_CREAT|O_RDWR如果文件不存在，则创建，否则以读写方式打开文件；
- O_CREAT|O_EXCL如果文件已存在，则调用失败，用来实现互斥；
- O_RDONLY、O_WRONLY、O_RDWR只能选择一个；
- O_CREAT | O_WRONLY|O_TRUNC组合，则与creat函数等价。

mode模式，即访问权限，包含如下选项：

（1）S_IRUSR，属主读权限；

（2）S_IWUSR，属主写权限；

（3）S_IXUSR，属主执行权限。

返回值 返回0表示正确，返回非零表示失败。

Example5.3.4-1： 打开文件

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
int main(int argc,char *argv[])
{
    int fd;
    int err=access(argv[1],0);
    printf("err=%d,errstr=%s\n",errno,strerror(errno));
    if (err==0){
//打开文件操作
        fd=open(argv[1],O_RDWR|O_SYNC, 0666);
    }
    else {
      //创建文件操作
        fd=creat(argv[1],0666);
    }
    if (fd==-1) {
      printf("open or creat  err \n");
      return 0;
    }
}
```

5.3.5关闭文件

表头文件

#include <unist.h>

定义函数

void close(int filedes);

参数：filedes为文件描述符。

Example5.3.5-1：当文件存在时，读取数据，否则创建文件并写入数据

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
int main(int argc,char **argv)          {
int fd,err=access(argv[1],0);  //检查文件是否存在
if (!err){    //若存在
   fd=open(argv[1],O_RDONLY);  //调用open函数，以只读方式打开文件
   char buf[4096];
   int  size=read(fd,buf,200); //调用read函数，将已打开的文件读取200字节数据，存入到buf中
 printf("--fd=%d,size=%d,buf=%s\n",fd,size,buf);  //输出fd、size以及buf中的数据
close(fd); //关闭文件
}
else {     //若不存在
fd=creat(argv[1],0777);   //调用create函数创建文件
char buf[256];
scanf("%s",buf);
int size=write(fd,buf,strlen(buf)); //向文件写入数据
close(fd);          //关闭文件
}
}
```

5.3.6文件删除

表头文件

```
#include <unistd.h>
```

定义函数

```
int unlink(const char *pathname);
```

参数：pathname为硬链接的各种名。

本质是解决硬链接，也就是删除节点数。

03

设备控制

设备控制，就是在设备驱动中对设备I/O进行管理。即对设备的一些特性进行控制，例如串口的传输波特率、马达的转速、混音器的音量、声卡的采样率等进行设置。

本节主要对ALSA声音编程进行简要介绍。

1.ALSA概述

ALSA是Advanced Linux Sound Architecture的缩写，目前已经成为了linux下的主流音频体系架构，提供了音频和MIDI的支持，替代了原先旧版本中的OSS（开发声音系统）。

在应用层，ALSA为我们提供了一套标准的API，应用程序只需要调用这些API就可完成对底层音频硬件设备的控制，譬如播放、录音等，这一套API称为alsa-lib。对应用程序提供了统一的API接口，这样可以隐藏驱动层的实现细节，简化应用程序实现难度、无需应用程序开发人员直接去读写音频设备节点。

2.sound设备节点

在Linux内核设备驱动层、基于ALSA音频驱动框架注册的sound设备会在/dev/snd目录下生成相应的设备节点文件，例如在终端进入/dev/snd目录并输入ls -l指令，可查询如下信息：

```
root@ubuntu:/home/linux# cd /dev/snd
root@ubuntu:/dev/snd# ls -l
total 0
drwxr-xr-x  2 root root       60 2月  10 14:56 by-path
crw-rw----+ 1 root audio 116,  2 2月  10 14:56 controlC0
crw-rw----+ 1 root audio 116,  6 2月  10 14:56 midiC0D0
crw-rw----+ 1 root audio 116,  4 2月  10 14:57 pcmC0D0c
crw-rw----+ 1 root audio 116,  3 2月  10 16:26 pcmC0D0p
crw-rw----+ 1 root audio 116,  5 2月  10 14:56 pcmC0D1p
crw-rw----+ 1 root audio 116,  1 2月  10 14:56 seq
crw-rw----+ 1 root audio 116, 33 2月  10 14:56 timer
```

（1）**controlC0：**用于声卡控制的设备节点，譬如通道选择、混音器、麦克风的控制等，C0表示声卡0（card0）；

（2）**pcmC0D0c：**用于录音的PCM设备节点。其中C0表示card0，也就是声卡0；而D0表示device 0，也就是设备0；最后一个字母c是capture的缩写，表示录音；所以pcmC0D0c便是系统的声卡0中的录音设备0；

（3）**pcmC0D0p：**用于播放（或叫放音、回放）的PCM设备节点。其中C0表示card0，也就是声卡0；而D0表示device 0，也就是设备0；最后一个字母p是playback的缩写，表示播放；所以pcmC0D0p便是系统的声卡0中的播放设备0；

（4）**pcmC0D1c：**用于录音的PCM设备节点。对应系统的声卡0中的录音设备1；

（5）**pcmC0D1p：**用于播放的PCM设备节点。对应系统的声卡0中的播放设备1。

（6）**timer：**定时器。

在Linux系统的/proc/asound目录下，有很多的文件，这些文件记录了系统中声卡相关的信息。

cards：

通过“cat /proc/asound/cards”命令、查看cards文件的内容，可列出系统中可用的、注册的声卡。

```
1. root@ubuntu:/dev/snd# cat /proc/asound/cards
2.  0 [AudioPCI       ]: ENS1371 - Ensoniq AudioPCI
3.                       Ensoniq AudioPCI ENS1371 at 0x2040, irq 16
```

系统中注册的所有声卡都会在/proc/asound/目录下形成相应的目录，该目录的命名方式为cardX（X表示声卡的编号），譬如上文中的card0；card0目录下记录了声卡0相关的信息，如声卡的名字以及声卡注册的PCM设备。

```
root@ubuntu:/proc/asound# ls
AudioPCI  card0  cards  devices  modules  oss  pcm  seq  timers  version
root@ubuntu:/dev/snd# cd /proc/asound/card0
root@ubuntu:/proc/asound/card0# ls
audiopci  codec97#0  id  midi0  pcm0c  pcm0p  pcm1p
```

devices：

devices目录列出系统中所有声卡注册的设备，包括control、pcm、timer、seq等等。

root@ubuntu:/proc/asound/card0# cat /proc/asound/devices

1: : sequencer

2: [0] : control

3: [0- 0]: digital audio playback

4: [0- 0]: digital audio capture

5: [0- 1]: digital audio playback

6: [0- 0]: raw midi

33: : timer

pcm：

pcm目录列出系统中的所有PCM设备，包括playback和capture。

1. root@ubuntu:/proc/asound/card0# cat /proc/asound/pcm
2. 00-00: ES1371/1 : ES1371 DAC2/ADC : playback 1 : capture 1
3. 00-01: ES1371/2 : ES1371 DAC1 : playback 1

3.控制采样设备

在alsa-lib应用编程中会涉及到一些概念，具体设计如下：

（1）样本长度（Sample）

样本是记录音频数据最基本的单元，样本长度就是采样位数，也称为位深度。

（2）声道数（channel）

分为单声道(Mono)和双声道/立体声(Stereo)。1表示单声道、2表示立体声。

（3）帧（frame）

帧记录了一个声音单元，其长度为样本长度与声道数的乘积，一段音频数据就是由苦干帧组成的。

（4）采样率（Sample rate）

也叫采样频率，是指每秒钟采样次数，该次数是针对帧而言。常见的采样率有：

- 8,000 Hz：电话所用采样率, 对于人的说话已经足够。
- 22,050 Hz：无线电广播所用采样率。
- 32,000 Hz：miniDV，数码视频 camcorder、DAT (LP mode)所用采样率。
- 44,100 Hz：音频CD，也常用于MPEG-1音频（VCD，SVCD，MP3）所用采样率。
- 47,250 Hz：PCM录音机所用采样率。
- ……

（5）交错模式（interleaved）

交错模式是一种音频数据的记录方式，分为交错模式和非交错模式。

（6）周期（period）

周期是音频设备处理（读、写）数据的单位，也就是说音频设备读写数据的单位是周期，每一次读或写一个周期的数据，一个周期包含若干个帧。

（6）缓冲区（buffer）

一个缓冲区包含若干个周期，所以buffer是由若干个周期所组成的一块空间。例如一个buffer包含4个周期、而一个周包含1024帧、一帧包含两个样本（左、右两个声道）。

04

Linux时间编程

时间的概念主要有世界标准时间和日历时间。

协调世界时（Coordinated Universal TimeUTC），又称为世界标准时间，也就是大家所熟知的格林威治标准时间（Greenwich MeanTime，GMT）。

日历时间（Calendar Time），是用“从一个标准时间点（如：1970年1月1日0点）到此时经过的秒数”来表示的时间。这是最基础的计量方式，有了这个基础数据，其它标准时间，本地时间便可轻松转化出来了。

5.5.1取得目前的时间

表头文件

#include<time.h>

定义函数

time_t time(time_t *t);

函数说明 此函数会返回从公元1970年1月1日（UTC时间）从0时0分0秒算起到现在所经过的秒数。如果t并非空指针的话，此函数也会将返回值存到t指针所指的内存。

返回值 成功则返回秒数，失败则返回((time_t)-1)值。

Example5.5.1-1:time函数使用方法

```
#include<time.h>  //头文件
#include <stdio.h>
int main(){
        int seconds= time((time_t*)NULL); //调用time_t time(time_t *t);函数， 计算秒数
        printf("%d\n",seconds);
}
```

5.5.2取得目前时间和日期

通常用户得到日历时间的秒数后可以将这些秒数转化为更容易接受的时间表示方式，这些表示时间的方式有格林威治时间、本地时间等。首先介绍获取格林威治时间的方法。

表头文件

#include<time.h>

定义函数

struct tm*gmtime(const time_t*timep);

函数说明 gmtime()将参数timep所指的time_t 结构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返回。

tm的结构定义为：

```
struct tm
{
    int tm_sec; //代表目前秒数，正常范围为0-59，但允许至61秒。
    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。
};
```

此函数返回的时间日期未经时区转换，而是UTC时间。

返回值 返回结构体tm代表目前UTC时间。

Example5.5.2-1:gmtime()函数使用方法

```
#include<time.h>
#include <stdio.h>
int main()
{
        char *wday[]={"Sun","Mon","Tue","Wed","Thu","Fri","Sat"}; //定义星期数组
        time_t timep;
        struct tm *p;
        time(&timep);
        p=gmtime(&timep);//调用gmtime()函数
        printf("%d-%d-%d\n",(1900+p->tm_year),(1+p->tm_mon),p->tm_mday);// 获 取
UTC时间
        printf("%s%d:%d:%d\n",wday[p->tm_wday],p->tm_hour,p->tm_min, p->tm_sec);
}
```

5.5.3取得当地目前时间和日期

表头文件

```
#include<time.h>
```

定义函数

```
struct tm *localtime(const time_t * timep);
```

函数说明 localtime()将参数timep所指的time_t结构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返回。此函数返回的时间日期已经转换成当地时区。

返回值 返回结构体tm代表目前的当地时间。

Example5.5.3-1:localtime函数使用方法

```
#include<time.h>
#include <stdio.h>
int main(){
    char *wday[]={"Sun","Mon","Tue","Wed","Thu","Fri","Sat"};//定义星期数组
    time_t timep;
    struct tm *p;
    time(&timep);
    p=localtime(&timep); //调用localtime函数
    printf("%d-%d-%d ",(1900+p->tm_year), (1+p->tm_mon),p->tm_mday);//获取本地时间
    printf("%s%d:%d:%d\n", wday[p->tm_wday], p->tm_hour, p->tm_min, p->tm_sec);
}
```

5.5.4将时间结构数据转换成经过的秒数

表头文件

#include<time.h>

定义函数

time_t mktime(strcut tm * timeptr);

函数说明 mktime()用来将参数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。

返回值 返回经过的秒数。

Example5.5.4-1:用time()取得时间（秒数），利用localtime()转换成结构tm，再利用mktine（）将结构 tm转换成原来的秒数。

```
#include<time.h>
#include <stdio.h>
int main(){
    time_t timep;
    struct tm *p;
    time(&timep);  //取得时间（秒数）
    printf("time(): %ld\n",timep);
    p=localtime(&timep); //获取当地时间和日期
    timep= mktime(p);//调用mktime()函数，将结构 tm转换成原来的秒数
    printf("time()->localtime()->mktime():%ld\n",timep);
}
```

5.5.5设置目前时间

表头文件

```
#include<sys/time.h>
#include<unistd.h>
```

定义函数

```
int settimeofday(const struct 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()。

5.5.6取得当前时间

表头文件

```
#include <sys/time.h>
#include <unistd.h>
```

定义函数

```
int gettimeofday ( struct timeval * 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*/
};
```

struct timeval结构类型提供了一个微秒级成员tv_usec，它的类型同样是一个整型类型。而gettimeofday函数的tz参数用于获取时区信息。

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*/
};
```

返回值 成功则返回0，失败返回-1。

Example5.5.6-1:gettimeofday函数使用方法

```
#include<stdio.h>
#include<sys/time.h>
#include<unistd.h>
int main()
{
    struct timeval tv;
    struct timezone tz;
    gettimeofday (&tv , &tz);
    printf("tv_sec; %ld\n", tv.tv_sec);//输出秒
    printf("tv_usec; %ld\n",tv.tv_usec);//输出微秒
    printf("tz_minuteswest; %d\n", tz.tz_minuteswest);//输出与Greenwich 时间差多少分钟
    printf("tz_dsttime, %d\n",tz.tz_dsttime);  //输出日光节约时间的状态
}
```

5.5.7将时间和日期以ASCII格式表示

利用函数gmtime()、localtime()可以将日历时间转化为格林威治时间和本地时间，虽然用户可通过结构体tm来获取这些时间值，但看起来还不方便，最好是将所有的信息，如年、月、日、星期、时、分、秒以字符串的形式显示出来。

表头文件

#include<time.h>

定义函数

char * asctime(const struct tm * timeptr);

函数说明 将tm格式的时间转化为字符串，若再调用相关的时间日期函数，此字符串可能会被破坏。此函数与ctime不同处在于传入的参数是不同的结构。

返回值 返回一字符串表示目前当地的时间日期。

例如: SAT Jul 30 08:43:03 2005。

该函数必须按照下面3个步骤来进行。

（1）使用函数time()来获取日历时间；

（2）使用函数gmtime()将日历时间转化为格林威治标准时间；

（3）使用函数asctime()将tm格式的时间转化为字符串。

Example5.5.7-1:asctime函数使用方法

```
#include <time.h>
#include<stdio.h>
int main()
{
    time_t timep;
    time (&timep);
    printf("%s",asctime(gmtime(&timep)));//将时间转化为字符串输出
}
```

5.5.8将时间和日期以字符串格式表示

表头文件

#include<time.h>

定义函数

char *ctime(const time_t *timep);

函数说明 将日历时间转化为本地时间的字符串形式，若再调用相关的时间日期函数，此字符串可能会被破坏。

返回值 返回一字符串表示目前当地的时间日期。

该函数必须按照下面2个步骤来进行。

（1）使用函数time()来获取日历时间；

（2）使用函数ctime()将日历时间直接转化为字符串。

Example5.5.8-1:ctime函数使用方法。

```
#include<time.h>
#include<stdio.h>
int main()
{
time_t timep;
time (&timep);
printf("%s",ctime(&timep));//将日历时间直接转化为字符串输出
}
```

Example5.5.8-2:写一段程序，实现本地时间和格林尼治时间转换。

```
#include <time.h>
#include <stdio.h>
int main(void){
    struct tm *local;
    time_t t;
    /* 获取日历时间 */
    t=time(NULL);
    /* 将日历时间转化为本地时间 */
    local=localtime(&t);
    /*打印当前的小时值*/
    printf("Local hour is: %d\n",local->tm_hour);
     /* 将日历时间转化为格林威治时间 */
    local=gmtime(&t);
    printf("UTC hour is: %d\n",local->tm_hour);
    return 0;
}
```

(1)文件I/O编程指的是什么，可以用那些方法实现？

(2)基于C语言的库函数文件编程和基于Linux的系统的文件编程区别是什么？

(3)什么是标准时间，什么是日历时间？

(4)编写写入数据程序，将一串字符串a~f写入到/tmp/test.txt中。

(5)编写读取文件程序，将/tmp/test.txt中内容读出。

(6)编写时间转换程序，将当前时间转换为格林威治时间。

(7)使用creat函数创建一个文件，当没有文件名输入时，提示用户输入一个文件名。

重庆理工大学/电气学院

CHONGQING UNIVERSITY OF TECHNOLOGY

嵌入式Linux系统开发教程

—基于ARM处理器通用平台（arm9-arm11-contexA系列）