

C語言操作時間函數，實現定時執行某個任務小程序

C語言與C++編程 今天

以下文章附帶一口Linux，作者土豆居士



一口Linux

15年嵌入式開發經驗古董級老鳥。曾任職中興通訊，某研究所，華清遠見教學總監。Linux驅動入門可以一起交流。

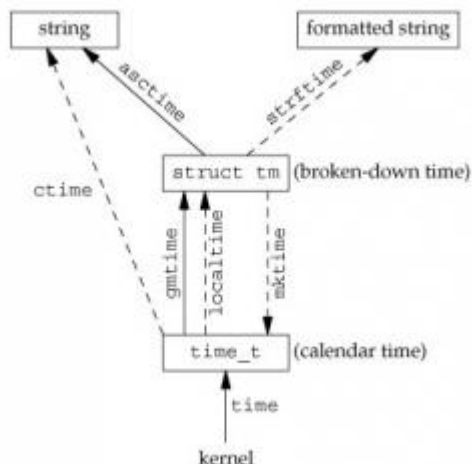


來自公眾號：**一口Linux**

鏈接：<https://blog.csdn.net/daocaokafei/article/details/108610289>

時間操作函數在實際項目開發中會經常用到，最近做項目也正好用到就正好順便整理一下。

時間概述



由上圖可知：

1. 通過系統調用函數`time ()`可以從內核獲得一個類型為`time_t`的1個值，該值叫calendar時間，即從1970年1月1日的UTC時間從0時0分0秒算到現在所經過的秒數。而該時間也用於紀念UNIX的誕生。
2. 函數`gmtime ()`，`localtime ()`可以將日曆時間轉換成`struct tm`結構體類型變量中。通過該結構體成員可以很方便的獲取當前的時間信息。我們也可以通過函數`mktime`變為類型結構體的變量轉變成calendar時間。

```
struct tm{
    int tm_sec; /*秒数*/
    int tm_min; /*分钟*/
    int tm_hour; /*小时*/
    int tm_mday; /*日期*/
    int tm_mon; /*月份*/
    int tm_year; /*从1990年算起至今的年数*/
    int tm_wday; /*星期*/
}
```

```
int tm_yday; /*从今年1月1日算起至今的天数*/  
int tm_isdst; /*日光节约时间的旗标*/  
};
```

3. asctime () 和ctime () 函數產生形式的26字節字符串，這與日期命令的系統轉換輸出形式類似：Tue Feb 10 18:27:38 2020 / n / 0。

4. strftime () 將一個struct tm結構格式化為一個字符串。

常用時間函數及模仿

1, 時間函數



头文件：time.h

函数定义：time_t time (time_t *t)

说明：

返回从1970年1月1日的UTC时间从0时0分0秒算起到现在所经过的秒数。


體現如下：



```
#include<stdio.h>  
#include<time.h>  
  
int main(){  
    time_t timep;
```

```
long seconds = time(&timep);
printf("%ld\n",seconds);
printf("%ld\n",timep);
return 0;
}
```

輸出：



```
1600177884
1600177884
```

有興趣的同學可以計算下，從1970年1月1日0時0分0秒到現在經歷了多少秒。

附：time_t 一路追蹤发现就是从long类型经过不断的typedef ,#define定义过来的。

2、ctime函数



定义：`char *ctime(const time_t *timep);`

说明：将参数所指的time_t结构中的信息转换成真实世界的时间日期表示方法，然后将结果以字符串形式返回。

注意这个是本地时间。

举例如下：



```
#include <stdio.h>
#include<time.h>
int main(void) {
    time_t timep;
```

```
time_t timep;  
  
time(&timep);  
printf("%s\n", ctime(&timep));  
  
return 0;  
}
```

輸出：

Tue Sep 15 06:53:02 2020

3、gmtime函数

定义：`struct tm *gmtime(const time_t *timep);`

说明：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未经时区转换，而是UTC

举例如下：

```
#include <stdio.h>  
#include <time.h>  
  
int main(void) {  
    char *wday[] = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};  
  
    time_t timep;  
    struct tm *p;
```

```
time(&timep);
p = gmtime(&timep);
printf("%d/%d/%d ", (1900+p->tm_year), (1+p->tm_mon), p->tm_mday);
printf("%s %d:%d:%d\n", wday[p->tm_wday], p->tm_hour, p->tm_min, p->tm_sec);
return 0;
}
```

输出:

2020/9/15 Tue 13:55:13

4、strftime函数

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

定义：

```
size_t strftime(char *s, size_t max, const char *format, const struct tm *tm);
```

说明：

类似于`snprintf`函数，我们可以根据`format`指向的格式字符串，将`struct tm`结构体中信息输出到`s`指针指向的字符串中，最多为`max`个字节。当然`s`指针指向的地址中，格式化字符串各种日期和时间的详细的确切表示方法有如下多种，我们可以根据需要来格式化各种各样的含时间字符串。

%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 年份的前两位数字

%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**，星期一是**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

返回值：

成功的话返回格式化之后s字符串的字节数，不包括null终止字符，但是返回的字符串包括null字节终止字符。否则返回**0**，s字符串的内容是未定义的。值得注意的是，

举例如下：

```
1 #include <stdio.h>
2 #include <time.h>
3
4 #define BUFLen 255
5 int main(int argc, char **argv)
6 {
7     time_t t = time( 0 );
8     char tmpBuf[BUFLen];
9
10    strftime(tmpBuf, BUFLen, "%Y%m%d%H%M%S", localtime(&t)); //format date a
11    printf("%s\n", tmpBuf);
12    return 0;
13 }
```

执行结果如下：

```
root@zh:~# ./run
20201003044421
```

输出结果表示YYYYmmDDHHMMSS

5、asctime函数

定义：


```
char *asctime(const struct tm *timeptr);
```

说明：

将参数timeptr所指的struct tm结构中的信息转换成真实时间所使用的时间日期表示方法，结果以字符串形态返回。与ctime()函数不同之处在于传入的参数是不同

返回值：

返回的也是UTC时间。

举例如下：

```
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
int main(void) {
    time_t timep;

    time(&timep);
    printf("%s\n",asctime(gmtime(&timep)));
    return EXIT_SUCCESS;
}
```

输出：

Tue Sep 15 13:56:26 2020

6、localhost函数

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

取得当地目前的时间和日期

举例如下:

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

int main(void) {
    char *wday[] = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};
    time_t timep;
    struct tm *p;

    time(&timep);
    p = localtime(&timep);
    printf("%d/%d/%d ", (1900+p->tm_year), (1+p->tm_mon), p->tm_mday);
    printf("%s %d:%d:%d\n", wday[p->tm_wday], p->tm_hour, p->tm_min, p->tm_sec);
    return EXIT_SUCCESS;
}
```

输出:

2020/9/15 Tue 6:57:10

7、mktime函数

定义：`time_t mktime(struct tm *timeptr);`

说明：

用来将参数`timeptr`所指的`tm`结构数据转换成从1970年1月1日的UTC时间从0时0分0秒算起到现在所经过的秒数。

举例如下：

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

int main(void) {
    time_t timep;
    struct tm *p;

    time(&timep);
    printf("time():%ld\n",timep);
    p = localtime(&timep);
    timep = mktime(p);
    printf("time()->localtime()->mktime():%ld\n",timep);
    return EXIT_SUCCESS;
}
```

输出：

```
time():1600178310
time()->localtime()->mktime():1600178310
```

8、gettimeofday函数



定义：

```
int gettimeofday(struct timeval *tv,struct timezone *tz);
```

说明：

把目前的时间由tv所指的结构返回，当地时区信息则放到有tz所指的结构中。

结构体timeval 定义如下：



```
struct timeval{  
    long tv_sec; /*秒*/  
    long tv_usec; /*微秒*/  
};
```

结构体timezone定义如下：



```
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*/  
}
```

举例如下：



```
#include <stdio.h>
```

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

int main(void) {
    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);
    return EXIT_SUCCESS;
}
```

输出:

```
tv_sec :1600178409
tv_usec: 543392
tz_minuteswest:420
tz_dsttime:0
```

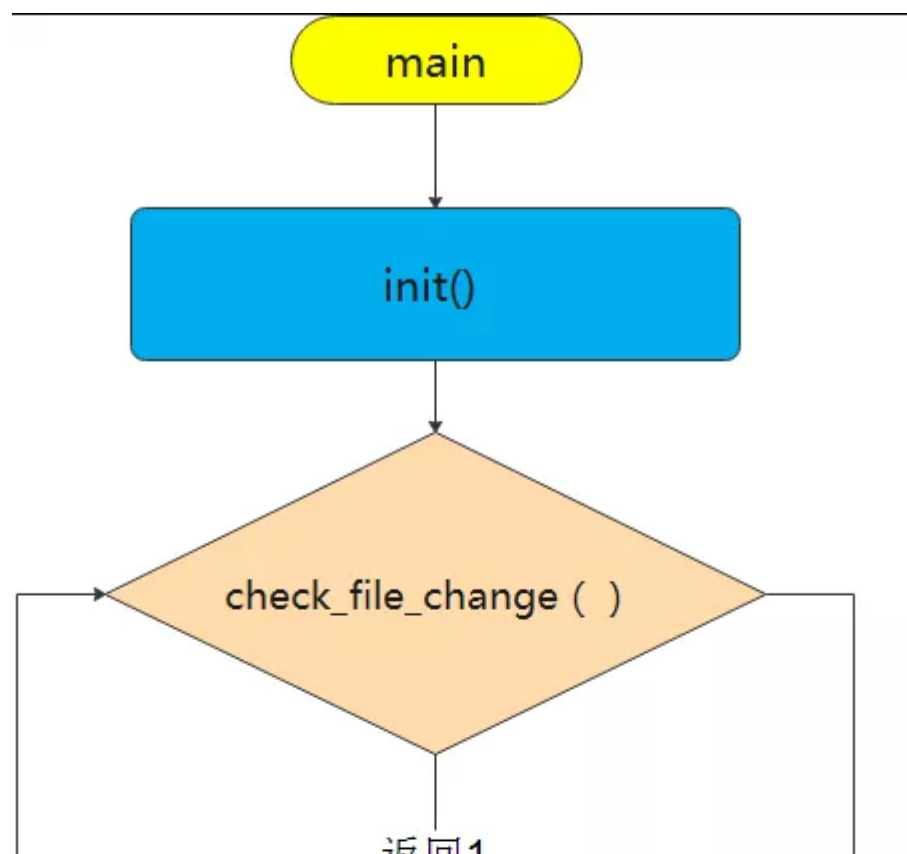
综合实验

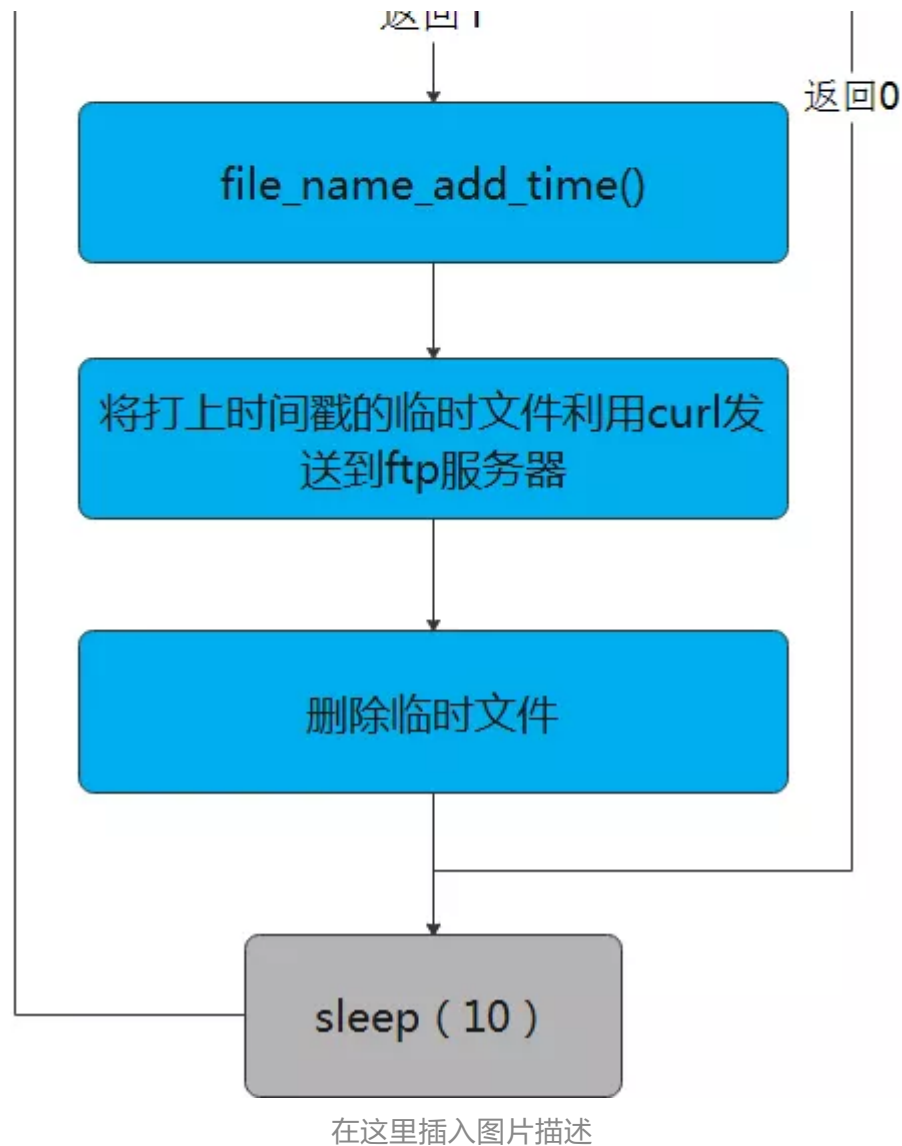
现在我们利用这些时间函数，来实现一个定时执行某个任务得功能。

功能

1. 程序运行时要记录当前日志文件的最后修改时间;
2. 每个10秒钟就检查下log文件是否被修改, 如果没有被修改就休眠10秒钟;
3. 如果log文件被修改了, 就将当前的日志文件拷贝成备份文件, 备份文件名字加上当前时间;
4. 通过curl发送给ftp服务器;
5. 删除备份文件, 重复步骤2。

程序流程图如下:





函数功能介绍

init()

首先记录当前log文件时间，并记录到全局变量 `last_mtime` 中。

check_file_change()读取文件最后修改时间，并和 `last_mtime` 进行比较，如果相同就返回0，不同就返回1。

file_name_add_time()将当前的日志文件拷贝成备份文件，备份文件名字加上当前时间。

stat()

得到对应文件的属性信息，存放到struct stat结构体变量中。

运行截图：

第一步：

```
root@ubuntu:/home/peng/driver/stat# ./a.out
init time:Wed Aug 26 01:09:04 2020

file not change
file not change
```

因为log文件没有被修改过，所以程序不会上传。

第二步：手动输入字符串 yikoulinux 到日志文件 t.log中。

```
root@ubuntu:/home/peng/driver/stat# echo yikoulinux > t.log
root@ubuntu:/home/peng/driver/stat#
```

第三步：因为文件发生了改变，所以打印“file updated”，同时可以看到curl上传文件的log信息。

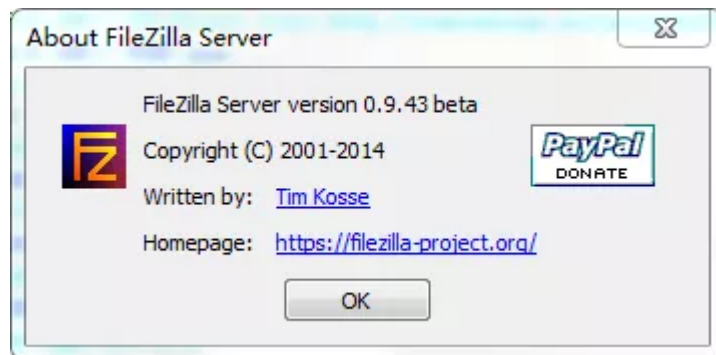
```
file not change
file not change
file updated
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   Dload  Upload   Total             Spent    Left     Speed
100    11    0    0  100    11      0    252  --:--:-- --:--:-- --:--:--    268
file not change
```

以下是FTP服务器的根目录，可以看到，上传的日志文件：`t-2020-7-26-1-19-45.log`。



【补充】

1. 配置信息，直接在代码中写死，通常应该从配置文件中读取，为方便读者阅读，本代码没有增加该功能；
2. FTP服务器搭建，本文没有说明，相关文件比较多，大家可以自行搜索，一口君用的是File zilla；



3. 通常这种需要长时间运行的程序，需要设置成守护进程，本文没有添加相应功能，读者可以自行搜索。如果强烈要求可以单开一篇详细介绍。
4. 代码中time的管理函数，请读者自行搜索相关文章。
5. curl也提供了相关的函数库curl.lib，如果来实现更灵活的功能可以使用对应的api。

6. 之所以先把文件拷贝成备份文件，主要是考虑其他模块随时可能修改日志文件，起到一定保护作用。

代码如下

代码如下：

```
/*
*****
Copyright (C) 公众号：一口Linux
*****
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

typedef struct stat ST;
unsigned long last_mtime;

/*用户名密码暂时写死，实际应该保存在配置文件*/
char name[32]="user";
char pass[32] ="123456";
char ip[32]      ="192.168.43.117";
char filename[32]="t.log";
char dstfile[256]  ={0};

int init(void)
{
    //准备结构体
}
```

```
ST status;

//调用stat函数
int res = stat(filename,&status);
if(-1 == res)
{
    perror("error:open file fail\n");
    return 0;
}
last_mtime = status.st_mtime;
printf("init time:%s \n",ctime(&last_mtime));
return 1;
}

int check_file_change(void)
{
    //准备结构体
    ST status;

    //调用stat函数
    int res = stat(filename,&status);
    if(-1 == res)
    {
        perror("error:open file fail\n");
        return 0;
    }
    // printf("old:%s new:%s",ctime(&last_mtime),ctime(&status.st_mtime));

    if(last_mtime == status.st_mtime)
    {
        printf("file not change\n");
        return 0;
    }
}
```

```
}else{
    printf("file updated\n");
    last_mtime = status.st_mtime;
    return 1;
}

}

void file_name_add_time(void)
{
    ST status;
    time_t t;
    struct tm *tblock;
    char cmd[1024]={0};

    t = time(NULL);
    tblock = localtime(&t);

    sprintf(dstfile,"t-%d-%d-%d-%d-%d.log",
        tblock->tm_year+1900,
        tblock->tm_mon,
        tblock->tm_mday,
        tblock->tm_hour,
        tblock->tm_min,
        tblock->tm_sec);
    sprintf(cmd,"cp %s %s",filename,dstfile);
    // printf("cdm=%s\n",cmd);
    system(cmd);
}

int main(void)
{
    char cmd[1024]={0};
```

```
init();  
while(1)  
{  
    if(check_file_change() == 1)  
    {  
        file_name_add_time();  
        sprintf(cmd,"curl -u %s:%s ftp://%s/ -T %s",name,pass,ip,dstfile);  
        // printf("cdm=%s\n",cmd);  
        system(cmd);  
        unlink(dstfile);  
    }  
    sleep(10);  
}  
}
```



-END-

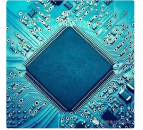
C语言与C++编程**分享C/C++技术文章**

[阅读原文](#)

喜欢此内容的人还喜欢

中国为何没有Intel和ARM这样的芯片巨头？该揭开真相了

嵌入式ARM



6 个“吓人”的 Linux 命令

Linux学习



数读 | 国产MCU真的可以做到替代吗？

與非網eefocus

