五级项目 3：获取时间写入文件

1 学时

- 1 学时

2 实验目的

- 理解、掌握、应用时间和文件相关函数的使用。

3 实验内容

- 获得当前系统时间，并以字符串形式保存到文件中。首先，需要调用 time 系统调用获取系统日历时间，再调用函数 localtime 将日历时间转换成需要的形式，最后调用函数 strftime 将结果格式化并将结果保存到文件中。

4 实验原理

（1）打开/创建文件 open

使用系统调用 open（）可以打开文件并得到一个文件描述符，该文件描述符用于对打开的文件进行访问。

表　系统调用 open/creat

项目	描述
头文件	#include <sys/types.h> #include <sys/stat.h> #include <fcntl.h>

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

功能	按照 flags 方式打开 pahtname 指定的文件，或者创建权限为 mode 的新文件 pathname
参数	pathname：要打开或者创建的文件名（包含路径） flags：打开方式（如表 4.5） mode：当要创建一个新文件时（flags 中包含 O_CREAT），mode 指定新创建文件的权限
返回值	int 型结果：当打开文件成功时，返回一个文件描述符；当打开文件失败时，返回-1，并且 errno 为错误码

成功地打开文件时，会得到一个文件描述符。文件描述符是唯一标识用户打开该文件的非负整数。这个整数其实是该文件在用户打开文件描述符表中表项的索引。

每个进程都有一个用户文件描述符表，用来记录该进程已经打开的文件。当一个进程通过 open()打开一个文件时，系统根据 pathname 找到该文件的 inode，校验该文件的属性信息是否可以打开。如果可以打开，则在该进程的用户文件描述符表中查找最小可用表项，并将

该表项的索引作为文件描述符返回。所以 open()函数总是得到一个最小可用的文件描述符。

当用户使用 open（）打开一个文件时，可以通过参数 flags 指定不同的打开方式。首先，打开方式 flags 必须取只读方式（O_RDONLY）、只写方式（O_WRONLY）和读写方式（O_RDWR）三者之一。除此之外，flags 还可以再增加其他的打开方式，如追加方式（O_APPEND）等等。打开方式 flags 取值如表。

表　打开文件方式 flags

flags	含义	备注
O_RDONLY	只读方式	三者必选其一，且只能选其一
O_WRONLY	只写方式	
O_RDWR	读写方式	
O_CREAT	如果 pathname 指定的文件不存在就创建该文件	
O_EXCL	与 O_CREAT 一起使用时，当 pathname 指定的文件已经存在，则 open（）函数失败并返回一个错误	
O_NOCTTY	如果 pathname 是终端设备，则不会把该终端设备当成进程控制终端	
O_TRUNC	如果 pathname 指定的文件是已经存在的普通文件（regular file），并且打开的方式是可写的（如 O_RDWR 或者 O_WRONLY），就将文件的长度截为 0。对于 FIFO 文件或者终端设备文件，该方式将被忽略	
O_APPEND	以追加的方式打开文件。在每一次调用 write（）写文件之前，文件指针将被自动置到文件末尾，该方式保证对文件的写都是在文件末尾进行追加	
O_NONBLOCK	以不可阻断的方式打开文件。无论有无数据读取或等待，都会立即返回进程	同 O_NDELAY
O_SYNC	文件以同步 I/O 的方式打开。这样每一次对文件的写操作之后进程都会阻塞直到数据物理地写到存储设备上	
O_NOFOLLOW	如果 pathname 是一个符号链接文件，则 open（）将执行失败。这是 FreeBSD 中的一个功能，后来加入到 Linux 2.1.126 中	
O_DIRECTORY	如果 pathname 不是一个目录，则 open（）将执行失败	

下表是文件打开方式 flags 几种常见的取值：

表 文件打开方式举例

flags 取值	含义
O_RDONLY	只读方式打开
O_RDWR|O_CREAT	如果文件存在就以读写方式打开，否则就创建文件
O_RDWR|O_CREAT|O_EXCL	如果文件不存在就创建，否则就返回错误
O_RDWR|O_CREAT|O_TRUNC	如果文件存在，则以读写方式打开，并将文件清空；否则创建该文件
O_WRONLY|O_APPEND	以只写方式打开文件，并且数据以追加的方式每次写入文件末尾
O_WRONLY|O_SYNC	以只写方式打开文件，并且每次写文件后等待数据

	真正写入磁盘后再返回进程

（2）读文件

当使用系统调用 open（）成功地打开或者新建一个文件后，可以得到一个文件描述符.通过该文件描述符可以对打开的文件进行访问，包括读和写。读文件使用系统调用 read（）。

表　系统调用 read

项目	描述
头文件	#include <unistd.h>
原型	ssize_t read(int fd, void *buf, size_t count);
功能	从文件描述符为 fd 的文件中读取 count 个字节的数据并放入 buf 缓冲区中 注意：从当前文件读写指针处开始读数据，读完后，当前文件读写指针也将改变成功读出的字节个数。
参数	fd：要读文件的文件描述符。在执行 read（）之前，应该执行 open/creat 打开或创建该文件，并得到该文件的文件描述符 buf：指向接收数据缓冲区的指针，读出的数据将存放在该缓冲区中 count：要读取数据的字节数
返回值	整型结果：执行成功时，返回的是实际读回数据的字节数；当失败时，返回-1，并且 errno 为错误码

当 read（）执行成功时，read（）返回的是实际读回数据的字节数，文件的读写指针会随之移动。此时会有以下几种情况：

返回值与 count 相等：实际读回的字节数与要读的字节数相等。读文件成功，并读回了想要字节数的数据。

返回值小于 count：实际读回的字节数小于要读的字节数。一般为读写指针接近文件末尾，文件剩余数据不够 count 个字节，此次 read 将剩余数据读出，并将实际读出字节数返回。

返回值为 0：表明从文件末尾读（所以读回来 0 个字节的数据）

（3）写文件

打开或者新建一个文件后，可以通过文件描述符对文件进行写操作，写文件使用系统调用 write（），如表所示。

表系统调用 write

项目	描述
头文件	#include <unistd.h>
原型	ssize_t write(int fd, const void *buf, size_t count);
功能	该函数将 buf 缓冲区中 count 个字节的数据写入文件描述符为 fd 的文件中 注意：写操作从文件读写指针处开始执行，写完后，当前文件读写指针也将改变成功写入的字节数。
参数	fd：要写文件的文件描述符。在执行 write（）之前，应该执行 open/creat 打开或创建该文件，并得到该文件的文件描述符 buf：指向要写数据缓冲区的指针，要写入文件的数据应先存放在该缓冲区中 count：要写入数据的字节数
返回值	整型结果：执行成功时，返回的是实际写入数据的字节数；当失败时，返回-1，并且 errno 为错误码

系统调用 write（）返回一个整型结果。

当 write（）执行成功时，write（）返回的是实际写入数据的字节数。文件的偏移量会随之移动。如果返回值为 0，表明没有写入任何数据（实际写入字节数为 0）。

当 write（）写文件失败时，返回-1，并且 errno 根据错误的原因被设置为相应的错误码。

（4）关闭文件

当一个打开的文件使用完之后，可以使用 close（）关闭该文件，如表所示。

表 系统调用 close

项目	描述
头文件	#include <unistd.h>
原型	int close(int fd);
功能	该函数关闭文件描述符为 fd 的文件。
参数	fd：要关闭的文件的文件描述符
返回值	如果成功的关闭，返回 0；如果关闭文件失败，则返回-1，并且 errno 为错误码

系统调用 close（）返回一个整型结果。如果成功关闭，返回 0；如果关闭文件失败，则返回-1，同时 errno 中设置错误码。

（5）strftime 函数

size_t strftime(char *str, size_t maxsize, const char *format, const struct tm *timeptr)

根据 format 中定义的格式化规则，格式化结构 timeptr 表示的时间，并把它存储在 str 中。

参数

str -- 这是指向目标数组的指针，用来复制产生的 C 字符串。

maxsize -- 这是被复制到 str 的最大字符数。

format -- 这是 C 字符串，包含了普通字符和特殊格式说明符的任何组合。这些格式说明符由函数替换为表示 tm 中所指定时间的相对应值。格式说明符是：

说明符	替换为	实例
%a	缩写的星期几名称	Sun
%A	完整的星期几名称	Sunday
%b	缩写的月份名称	Mar
%B	完整的月份名称	March
%c	日期和时间表示法	Sun Aug 19 02:56:02 2012
%d	一月中的第几天（01-31）	19

%H	24 小时格式的小时（00-23）	14
%I	12 小时格式的小时（01-12）	05
%j	一年中的第几天（001-366）	231
%m	十进制数表示的月份（01-12）	08
%M	分（00-59）	55
%p	AM 或 PM 名称	PM
%S	秒（00-61）	02
%U	一年中的第几周，以第一个星期日作为第一周的第一天（00-53）	33
%w	十进制数表示的星期几，星期日表示为 0（0-6）	4
%W	一年中的第几周，以第一个星期一作为第一周的第一天（00-53）	34
%x	日期表示法	08/19/12
%X	时间表示法	02:50:06
%y	年份，最后两个数字（00-99）	01
%Y	年份	2012
%Z	时区的名称或缩写	CDT
%%	一个 % 符号	%

timeptr -- 这是指向 tm 结构的指针，该结构包含了一个呗分解为以下各部分的日历时间：

```
struct tm {
    int tm_sec;         /* 秒，范围从 0 到 59                */
    int tm_min;         /* 分，范围从 0 到 59                */
    int tm_hour;        /* 小时，范围从 0 到 23              */
    int tm_mday;        /* 一月中的第几天，范围从 1 到 31    */
    int tm_mon;         /* 月份，范围从 0 到 11              */
    int tm_year;        /* 自 1900 起的年数                  */
    int tm_wday;        /* 一周中的第几天，范围从 0 到 6     */
    int tm_yday;        /* 一年中的第几天，范围从 0 到 365   */
    int tm_isdst;       /* 夏令时                            */
```

};

返回值：如果产生的 C 字符串小于 size 个字符（包括空结束字符），则会返回复制到 str 中的字符总数（不包括空结束字符），否则返回零。

5 预习要求和技术准备工作

- 掌握 Linux 基本操作
- 掌握 C 语言开发工具的使用
- 掌握时间函数和文件操作函数的使用

6 实验环境

- PC 机
- 在 Windows 环境中的 VMware 虚拟机上运行 Ubuntu 操作系统或者独立的 Ubuntu 操作系统
- 基于 Linux 的 vi 编辑器和 gcc 编译器

7 实验设计及操作步骤

7.1 以 root 身份登录系统，在/home 目录中创建目录 exp53

```
cd  /home
mkdir  exp53
```

7.2 进入刚创建的目录

```
cd  exp53
```

7.3 使用 vi 编辑文件，文件名是 timetest.c

7.4 编写程序，实现要求的功能。

★编程思路

```
main(int argc,char *argv[])
{
    ① 获取当前时间
    ② 将当前时间利用 localtime 转换为本地时间
    ③ 利用格式化函数 strftime 转换为字符串
    ④ 打开文件，将字符串写入文件后关闭文件。
}
```

7.5 编译可执行文件。

```
# gcc  -o  timetest  timetest.c
```

7.6 运行程序

```
# ./timetest
```

查看 out 文件内容：

more out

结果类似：

```
Thu Jul 22 16:53:09 2021
```

8 实验报告提交要求：

将实验操作每个步骤中的命令、源程序以及截图写入实验报告，实验报告命名为“学号姓名-实验 53.doc”，交给指定人员。

9 项目思考

本例是获取时间写入文件，可以考虑根据用户输入修改系统时间是如何实现的。