NAME

ctime, ctime_r, localtime, localtime_r, gmtime, gmtime_r, asctime, asctime_r, tzset, tzsetwall – convert date and time to string

SYNOPSIS

#include <time.h>

char *ctime(const time_t ***clock*);**

char *ctime_r(const time_t ***clock,* **char ***buf,* **int *buflen***);**

struct tm *localtime(const time_t ***clock*);**

struct tm *localtime_r(const time_t ***clock,* **struct tm ***res*);**

struct tm *gmtime(const time_t ***clock*);**

struct tm *gmtime_r(const time_t ***clock,* **struct tm ***res*);**

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

char *asctime_r(const struct tm ***tm,*char ***buf,* **int** *buflen***);**

extern time_t timezone, altzone;
extern int daylight;
extern char *tzname[2];

void tzset(void);

void tzsetwall(void);

POSIX

cc [*flag* **...]** *file* **... –D_POSIX_PTHREAD_SEMANTICS [** *library* **...]**

char *ctime_r(const time_t ***clock,* **char ***buf,*);**

char *asctime_r(const struct tm ***tm,*char ***buf,***);**

MT-LEVEL

See the **NOTES** section of this page.

DESCRIPTION

ctime( ), **localtime( )**, and **gmtime( )** accept arguments of type **time_t**, pointed to by **clock( )**, representing the time in seconds since 00:00:00 UTC, January 1, 1970. **ctime( )** returns a pointer to a 26-character string as shown below. Time zone and daylight savings corrections are made before string generation. The fields are constant width:

Fri Sep 13 00:00:00 1986\n\0

ctime_r( ) has the same functionality as **ctime( )** except that the caller must supply a buffer *buf* with length *buflen* to store the result; *buf* must be at least 26 bytes. The POSIX **ctime_r( )** routine does not take a *buflen* parameter.

localtime( ) and **gmtime( )** return pointers to **tm** structures (see below). **localtime( )** corrects for the main time zone and possible alternate (''daylight savings'') time zone; **gmtime( )** converts directly to Coordinated Universal Time (UTC), which is what the UNIX system uses internally.

localtime_r( ) and **gmtime_r( )** have the same functionality as **localtime( )** and **gmtime( )** respectively, except that the caller must supply a buffer *res* to store the result.

asctime( ) converts a **tm** structure to a 26-character string, as shown in the above example, and returns a pointer to the string.

asctime_r( ) has the same functionality as **asctime( )** except that the caller must supply a buffer *buf* with length *buflen* for the result to be stored. *buf* must be at least 26 bytes. The POSIX **asctime_r( )** routine does not take a *buflen* parameter. **asctime_r( )** returns a pointer to *buf* upon success. In case of failure, **NULL** is returned and **errno** is set.

Declarations of all the functions and externals, and the **tm** structure, are in the **time.h** header. The members of the **tm** structure are:

```
int     tm_sec;         /* seconds after the minute — [0, 61] */
                  /* for leap seconds */
int     tm_min;         /* minutes after the hour — [0, 59] */
int     tm_hour;        /* hour since midnight — [0, 23] */
int     tm_mday;        /* day of the month — [1, 31] */
int     tm_mon;         /* months since January — [0, 11] */
int     tm_year;        /* years since 1900 */
int     tm_wday;        /* days since Sunday — [0, 6] */
int     tm_yday;        /* days since January 1 — [0, 365] */
int     tm_isdst;       /* flag for alternate daylight savings time */
```

The value of **tm_isdst** is positive if daylight savings time is in effect, zero if daylight savings time is not in effect, and negative if the information is not available. (Previously, the value of **tm_isdst** was defined as non-zero if daylight savings was in effect.)

The external **time_t** variable **altzone** contains the difference, in seconds, between Coordinated Universal Time and the alternate time zone. The external variable **timezone** contains the difference, in seconds, between UTC and local standard time. The external variable **daylight** indicates whether time should reflect daylight savings time. Both **timezone** and **altzone** default to 0 (UTC). The external variable **daylight** is non-zero if an alternate time zone exists. The time zone names are contained in the external variable **tzname**, which by default is set to:

```
char *tzname[2] = { "GMT", "" };
```

These functions know about the peculiarities of this conversion for various time periods for the U.S. (specifically, the years 1974, 1975, and 1987). They will handle the new daylight savings time starting with the first Sunday in April, 1987.

tzset( ) uses the contents of the environment variable **TZ** to override the value of the different external variables. The function **tzset( )** is called by **asctime( )** and may also be called by the user. See **environ**(5) for a description of the **TZ** environment variable.

Starting and ending times are relative to the current local time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year will be used and the time will be 2 AM. If the start and end dates are provided but the time is not provided, the time will be 2 AM. The effects of **tzset( )** change the values of the external variables **timezone**, **altzone**, **daylight**, and **tzname**.

Note that in most installations, **TZ** is set to the correct value by default when the user logs on, using the local **/etc/default/init** file (see **TIMEZONE**(4)).

tzsetwall( ) sets things up so that **localtime( )** returns the best available approximation of local wall clock time.

LC_TIME determines how these functions handle date and time formats. In the "C" locale, date and time handling follow the U.S. rules.

ERRORS

ctime_r( ) and **asctime_r( )** will fail if the following is true:

ERANGE length of the buffer supplied by caller is not large enough to store the result.

EXAMPLES

tzset( ) scans the contents of the environment variable and assigns the different fields to the respective variable. For example, the most complete setting for New Jersey in 1986 could be

EST5EDT4,116/2:00:00,298/2:00:00

or simply

EST5EDT

An example of a southern hemisphere setting such as the Cook Islands could be

KDT9:30KST10:00,63/5:00,302/20:00

In the longer version of the New Jersey example of **TZ, tzname**[*0*] is EST, **timezone** will be set to 5∗60∗60, **tzname**[*1*] is EDT, **altzone** will be set to 4∗60∗60, the starting date of the alternate time zone is the 117th day at 2 AM, the ending date of the alternate time zone is the 299th day at 2 AM (using zero-based Julian days), and **daylight** will be set positive. Starting and ending times are relative to the current local time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year will be used and the time will be 2 AM. If the start and end dates are provided but the time is not provided, the time will be 2 AM. The effects of **tzset( )** are thus to change the values of the external variables **timezone**, **altzone**, **daylight**, and **tzname**. **ctime( )**, **localtime( )**, **mktime( )**, and **strftime( )** will also update these external variables as if they had called **tzset( )** at the time specified by the **time_t** or **struct tm** value that they are converting.

FILES

/usr/lib/locale/*locale***/LC_TIME/time**
file containing locale specific date and time information

SEE ALSO

time(2), **getenv**(3C), **mktime**(3C), **printf**(3S), **putenv**(3C), **setlocale**(3C), **strftime**(3C), **TIMEZONE**(4), **environ**(5)

NOTES

When compiling multi-thread applications, the **_REENTRANT** flag must be defined on the compile line. This flag should only be used in multi-thread applications.

The return values for **ctime( )**, **localtime( )**, and **gmtime( )** point to static data whose content is overwritten by each call.

Setting the time during the interval of change from **timezone** to **altzone** or vice versa can produce unpredictable results. The system administrator must change the Julian start and end days annually.

asctime( ), **ctime( )**, **gmtime( )** and **localtime( )** are unsafe in multi-thread applications. **asctime_r( )**, **ctime_r( )**, **gmtime_r( )** and **localtime_r( )** are MT-Safe, and should be used instead. **tzset( )** and **tzsetwall( )** are unsafe in multi-thread applications.

All interfaces on this page with an _r suffix are as specified in POSIX 1003.1c Draft #10.