NAME

asctime, **asctime_r**, **ctime**, **ctime_r**, **difftime**, **gmtime**, **gmtime_r**, **localtime**, **localtime_r**, **mktime**, **timegm** - transform binary date and time values

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <time.h>

*extern char *tzname[2]*;

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

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

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

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

double
difftime(*time_t time1, time_t time0*);

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

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

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

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

time_t

mktime(*struct tm *tm*);

time_t
timegm(*struct tm *tm*);

DESCRIPTION

The **ctime**(), **gmtime**(), and **localtime**() functions all take as argument a pointer to a time value representing the time in seconds since the Epoch (00:00:00 UTC on January 1, 1970; see time(3)).

The **localtime**() function converts the time value pointed to by *clock*, and returns a pointer to a *struct tm* (described below) which contains the broken-out time information for the value after adjusting for the current time zone (see tzset(3)). When the specified time translates to a year that will not fit in an *int*, **localtime**() returns NULL. The **localtime**() function uses tzset(3) to initialize time conversion information if tzset(3) has not already been called by the process.

After filling in the *struct tm*, **localtime**() sets the *tm_isdst*'th element of *tzname* to a pointer to an ASCII string that is the time zone abbreviation to be used with **localtime**()'s return value.

The **gmtime**() function similarly converts the time value, but without any time zone adjustment, and returns a pointer to a *struct tm*.

The **ctime**() function adjusts the time value for the current time zone in the same manner as **localtime**(), and returns a pointer to a 26-character string of the form:

```
Thu Nov 24 18:22:48 1986\n\0
```

All the fields have constant width.

The **asctime**() function converts the broken down time in the *struct tm* pointed to by *tm* to the form shown in the example above.

The **ctime_r**() and **asctime_r**() functions provide the same functionality as **ctime**() and **asctime**() except the caller must provide the output buffer *buf*, which must be at least 26 characters long, to store the result in. The **localtime_r**() and **gmtime_r**() functions provide the same functionality as **localtime**() and **gmtime**() respectively, except the caller must provide the output buffer *result*.

The functions **mktime**() and **timegm**() convert the broken-down time in the *struct tm* pointed to by *tm* into a time value with the same encoding as that of the values returned by the time(3) function (that is, seconds from the Epoch, UTC). The **mktime**() function interprets the input structure according to the current timezone setting (see tzset(3)) while the **timegm**() function interprets the input structure as

representing Universal Coordinated Time (UTC).

The original values of the *tm_wday* and *tm_yday* components of the structure are ignored, and the original values of the other components are not restricted to their normal ranges, and will be normalized if needed. For example, October 40 is changed into November 9, a *tm_hour* of -1 means 1 hour before midnight, *tm_mday* of 0 means the day preceding the current month, and *tm_mon* of -2 means 2 months before January of *tm_year*. (A positive or zero value for *tm_isdst* causes **mktime**() to presume initially that summer time (for example, Daylight Saving Time) is or is not in effect for the specified time, respectively. A negative value for *tm_isdst* causes the **mktime**() function to attempt to guess whether summer time is in effect for the specified time. The *tm_isdst* and *tm_gmtoff* members are forced to zero by **timegm**().)

On successful completion, the values of the *tm_wday* and *tm_yday* components of the structure are set appropriately, and the other components are set to represent the specified calendar time, but with their values forced to their normal ranges; the final value of *tm_mday* is not set until *tm_mon* and *tm_year* are determined. The **mktime**() function returns the specified calendar time; if the calendar time cannot be represented, it returns -1 and sets errno(3) to an appropriate value.

Note that -1 is a valid result (representing one second before midnight UTC on the evening of 31 December 1969), so this cannot be relied upon to indicate success or failure; instead, *tm_wday* and / or *tm_yday* should be set to an out-of-bounds value (e.g. -1) prior to calling **mktime**() or **timegm**() and checked after the call.

The **difftime**() function returns the difference in seconds between two time values, *time1 - time0*.

External declarations as well as the definition of *struct tm* are in the <*time.h*> header. The *tm* structure includes at least the following fields:

```
int tm_sec;     /* seconds (0 - 60) */
int tm_min;     /* minutes (0 - 59) */
int tm_hour;    /* hours (0 - 23) */
int tm_mday;    /* day of month (1 - 31) */
int tm_mon;     /* month of year (0 - 11) */
int tm_year;    /* year - 1900 */
int tm_wday;    /* day of week (Sunday = 0) */
int tm_yday;    /* day of year (0 - 365) */
int tm_isdst;   /* is summer time in effect? */
char *tm_zone;  /* abbreviation of timezone name */
long tm_gmtoff; /* offset from UTC in seconds */
```

The *tm_isdst* field is non-zero if summer time is in effect.

The *tm_gmtoff* field is the offset in seconds of the time represented from UTC, with positive values indicating a time zone ahead of UTC (east of the Prime Meridian).

SEE ALSO

date(1), clock_gettime(2), gettimeofday(2), getenv(3), time(3), tzset(3), tzfile(5)

STANDARDS

The **asctime**(), **ctime**(), **difftime**(), **gmtime**(), **localtime**(), and **mktime**() functions conform to ISO/IEC 9899:1990 ("ISO C90"), and conform to ISO/IEC 9945-1:1996 ("POSIX.1") provided the selected local timezone does not contain a leap-second table (see zic(8)).

The **asctime_r**(), **ctime_r**(), **gmtime_r**(), and **localtime_r**() functions are expected to conform to ISO/IEC 9945-1:1996 ("POSIX.1") (again provided the selected local timezone does not contain a leap-second table).

The **timegm**() function is not specified by any standard; its function cannot be completely emulated using the standard functions described above.

HISTORY

This manual page is derived from the time package contributed to Berkeley by Arthur Olson and which appeared in 4.3BSD.

The functions **asctime**(), **gmtime**(), and **localtime**() first appeared in Version 5 AT&T UNIX, **difftime**() and **mktime**() in 4.3BSD-Reno, and **timegm**() and **timelocal**() in SunOS 4.0.

The **asctime_r**(), **ctime_r**(), **gmtime_r**() and **localtime_r**() functions have been available since FreeBSD 8.0.

BUGS

Except for **difftime**(), **mktime**(), and the **_r**() variants of the other functions, these functions leave their result in an internal static object and return a pointer to that object. Subsequent calls to these function will modify the same object.

The C Standard provides no mechanism for a program to modify its current local timezone setting, and the POSIX-standard method is not reentrant. (However, thread-safe implementations are provided in the POSIX threaded environment.)

The *tm_zone* field of a returned *tm* structure points to a static array of characters, which will also be

overwritten by any subsequent calls (as well as by subsequent calls to tzset(3)).

Use of the external variable *tzname* is discouraged; the *tm_zone* entry in the tm structure is preferred.