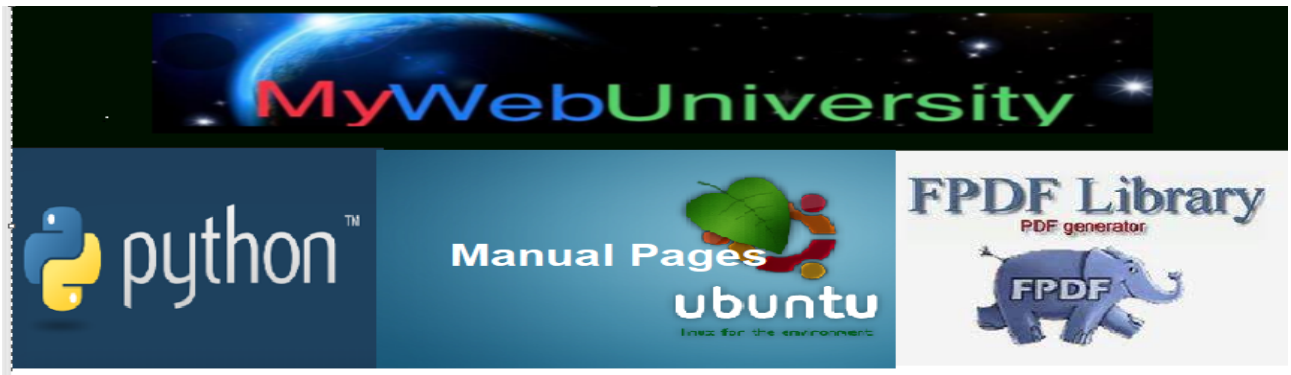

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'localtime.3'

$ man localtime.3

ctime(3) Library Functions Manual ctime(3)

NAME

asctime, ctime, gmtime, localtime, mktime, asctime_r, ctime_r, gmtime_r, localtime_r - transform date and time to broken-down time or ASCII

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <time.h>
char *asctime(const struct tm *tm);
char *asctime_r(const struct tm *restrict tm,
                char buf[restrict 26]);
char *ctime(const time_t *timep);
char *ctime_r(const time_t *restrict timep,
                char buf[restrict 26]);
struct tm *gmtime(const time_t *timep);
struct tm *gmtime_r(const time_t *restrict timep,
                struct tm *restrict result);
struct tm *localtime(const time_t *timep);
struct tm *localtime_r(const time_t *restrict timep,
                struct tm *restrict result);
time_t mktime(struct tm *tm);
```

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

asctime_r(), ctime_r(), gmtime_r(), localtime_r():

_POSIX_C_SOURCE

|| /* glibc <= 2.19: */ _BSD_SOURCE || _SVID_SOURCE

DESCRIPTION

The ctime(), gmtime(), and localtime() functions all take an argument of data type time_t, which represents calendar time. When interpreted as an absolute time value, it represents the number of seconds elapsed since the Epoch, 1970-01-01 00:00:00 +0000 (UTC).

The asctime() and mktime() functions both take an argument representing broken-down time, which is a representation separated into year, month, day, and so on.

Broken-down time is stored in the structure tm, described in tm(3type).

The call ctime(t) is equivalent to asctime(localtime(t)). It converts the calendar time t into a null-terminated string of the form

"Wed Jun 30 21:49:08 1993\n"

The abbreviations for the days of the week are "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", and "Sat". The abbreviations for the months are "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", and "Dec". The return value points to a statically allo? cated string which might be overwritten by subsequent calls to any of the date and time functions. The function also sets the external variables tzname, timezone, and daylight as if it called tzset(3). The reentrant version ctime_r() does the same, but stores the string in a user-supplied buffer which should have room for at least 26 bytes. It need not set tz? name, timezone, and daylight.

The gmtime() function converts the calendar time timep to broken-down time representation, expressed in Coordinated Universal Time (UTC). It may return NULL when the year does not fit into an integer. The return value points to a statically allocated struct which might be overwritten by subsequent calls to any of the date and time functions. The gmtime_r() function does the same, but stores the data in a user-supplied struct.

The localtime() function converts the calendar time timep to broken-down time representa? tion, expressed relative to the user's specified timezone. The function also sets the ex? ternal variables tzname, timezone, and daylight as if it called tzset(3). The return value points to a statically allocated struct which might be overwritten by subsequent calls to any of the date and time functions. The localtime_r() function does the same, but stores the data in a user-supplied struct. It need not set tzname, timezone, and daylight.

The asctime() function converts the broken-down time value tm into a null-terminated string with the same format as ctime(). The return value points to a statically allocated string

which might be overwritten by subsequent calls to any of the date and time functions. The asctime_r() function does the same, but stores the string in a user-supplied buffer which should have room for at least 26 bytes.

The mktime() function converts a broken-down time structure, expressed as local time, to calendar time representation. The function ignores the values supplied by the caller in the tm_wday and tm_yday fields. The value specified in the tm_isdst field informs mktime() whether or not daylight saving time (DST) is in effect for the time supplied in the tm structure: a positive value means DST is in effect; zero means that DST is not in effect; and a negative value means that mktime() should (use timezone information and system data? bases to) attempt to determine whether DST is in effect at the specified time. See timegm(3) for a UTC equivalent of this function.

The mktime() function modifies the fields of the tm structure as follows: tm_wday and tm_yday are set to values determined from the contents of the other fields; if structure members are outside their valid interval, they will be normalized (so that, for example, 40 October is changed into 9 November); tm_isdst is set (regardless of its initial value) to a positive value or to 0, respectively, to indicate whether DST is or is not in effect at the specified time. The function also sets the external variables tzname, timezone, and day? light as if it called tzset(3).

If the specified broken-down time cannot be represented as calendar time (seconds since the Epoch), mktime() returns (time_t) -1 and does not alter the members of the broken-down time structure.

RETURN VALUE

On success, gmtime() and localtime() return a pointer to a struct tm.

On success, gmtime_r() and localtime_r() return the address of the structure pointed to by result.

On success, asctime() and ctime() return a pointer to a string.

On success, asctime_r() and ctime_r() return a pointer to the string pointed to by buf.

On success, mktime() returns the calendar time (seconds since the Epoch), expressed as a value of type time_t.

On error, mktime() returns the value (time_t) -1, and leaves the tm->tm_wday member unmodi? fied. The remaining functions return NULL on error. On error, errno is set to indicate the error.

ERRORS

EOVERFLOW

The result cannot be represented.

ATTRIBUTES

For an explanation of the terms used in this section, see attributes(7).

Interface	Attribute	Value
asctime()	Thread safety	MT-Unsafe race:asctime locale
asctime_r()	Thread safety	MT-Safe locale
ctime()	Thread safety	MT-Unsafe race:tmbuf race:asctime env locale
ctime_r(), gmtime_r(), localtime_r(), mktime()	Thread safety	MT-Safe env locale
gmtime(), localtime()	Thread safety	MT-Unsafe race:tmbuf env locale

VERSIONS

POSIX doesn't specify the parameters of ctime_r() to be restrict; that is specific to glibc.

In many implementations, including glibc, a 0 in tm_mday is interpreted as meaning the last day of the preceding month.

According to POSIX.1, localtime() is required to behave as though tzset(3) was called, while localtime_r() does not have this requirement. For portable code, tzset(3) should be called before localtime_r().

STANDARDS

asctime()

ctime()

gmtime()

localtime()

mktime()

C23, POSIX.1-2024.

gmtime_r()

localtime_r()

POSIX.1-2024.

asctime_r()

ctime_r()

None.

HISTORY

gmtime()

localtime()

mktime()

C89, POSIX.1-1988.

asctime()

ctime()

C89, POSIX.1-1988. Marked obsolescent in C23 and in POSIX.1-2008 (recommending strf?
time(3)).

gmtime_r()

localtime_r()

POSIX.1-1996.

asctime_r()

ctime_r()

POSIX.1-1996. Marked obsolescent in POSIX.1-2008. Removed in POSIX.1-2024 (recom?
mending strftime(3)).

CAVEATS

Thread safety

The four functions asctime(), ctime(), gmtime(), and localtime() return a pointer to static data and hence are not thread-safe. The thread-safe versions, asctime_r(), ctime_r(), gm?
time_r(), and localtime_r(), are specified by SUSv2.

POSIX.1 says: "The asctime(), ctime(), gmtime(), and localtime() functions shall return val?
ues in one of two static objects: a broken-down time structure and an array of type char. Execution of any of the functions that return a pointer to one of these object types may overwrite the information in any object of the same type pointed to by the value returned from any previous call to any of them." This can occur in the glibc implementation.

mktime()

(time_t) -1 can represent a valid time (one second before the Epoch). To determine whether

mktime() failed, one must use the tm->tm_wday field. See the example program in EXAMPLES.

The handling of a non-negative tm_isdst in mktime() is poorly specified, and passing a value that is incorrect for the time specified yields unspecified results. Since mktime() is one of the few functions that knows when DST is in effect, providing a correct value may be dif? ficult. One workaround for this is to call mktime() twice, once with tm_isdst set to zero, and once with tm_isdst set to a positive value, and discarding the results from the call that changes it. If neither call changes tm_isdst then the time specified probably happens during a fall-back period where DST begins or ends, and both results are valid but represent two different times. If both calls change it, that could indicate a fall-forward transi? tion, or some other reason why the time specified does not exist.

The specification of time zones and daylight saving time are up to regional governments, change often, and may include discontinuities beyond mktime's ability to document a result. For example, a change in the timezone definition may cause a clock time to be repeated or skipped without a corresponding DST change.

EXAMPLES

The program below defines a wrapper that allows detecting invalid and ambiguous times, with EINVAL and ENOTUNIQ, respectively.

The following shell session shows sample runs of the program:

```
$ TZ=UTC ./a.out 1969 12 31 23 59 59 0;
-1
$
$ export TZ=Europe/Madrid;
$
$ ./a.out 2147483647 2147483647 00 00 00 00 -1;
a.out: mktime: Value too large for defined data type
$
$ ./a.out 2024 08 23 00 17 53 -1;
1724365073
$ ./a.out 2024 08 23 00 17 53 0;
a.out: my_mktime: Invalid argument
1724368673
$ ./a.out 2024 08 23 00 17 53 1;
1724365073
```

```
$
$ ./a.out 2024 02 23 00 17 53 -1;
1708643873
$ ./a.out 2024 02 23 00 17 53 0;
1708643873
$ ./a.out 2024 02 23 00 17 53 1;
a.out: my_mktime: Invalid argument
1708640273
$
$ ./a.out 2023 03 26 02 17 53 -1;
a.out: my_mktime: Invalid argument
1679793473
$
$ ./a.out 2023 10 29 02 17 53 -1;
a.out: my_mktime: Name not unique on network
1698542273
$ ./a.out 2023 10 29 02 17 53 0;
1698542273
$ ./a.out 2023 10 29 02 17 53 1;
1698538673
$
$ ./a.out 2023 02 29 12 00 00 -1;
a.out: my_mktime: Invalid argument
1677668400
```

Program source: mktime.c

```
#include <err.h>
#include <errno.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define is_signed(T)  ((T) -1 < 1)
```

```
static time_t my_mktime(struct tm *tp);

int

main(int argc, char *argv[])

{

    char       **p;

    time_t     t;

    struct tm  tm;

    if (argc != 8) {

        fprintf(stderr, "Usage: %s yyyy mm dd HH MM SS isdst\n", argv[0]);

        exit(EXIT_FAILURE);

    }

    p = &argv[1];

    tm.tm_year  = atoi(*p++) - 1900;

    tm.tm_mon   = atoi(*p++) - 1;

    tm.tm_mday  = atoi(*p++);

    tm.tm_hour  = atoi(*p++);

    tm.tm_min   = atoi(*p++);

    tm.tm_sec   = atoi(*p++);

    tm.tm_isdst = atoi(*p++);

    errno = 0;

    tm.tm_wday = -1;

    t = my_mktime(&tm);

    if (tm.tm_wday == -1)

        err(EXIT_FAILURE, "mktime");

    if (errno == EINVAL || errno == ENOTUNIQ)

        warn("my_mktime");

    if (is_signed(time_t))

        printf("%jd\n", (intmax_t) t);

    else

        printf("%ju\n", (uintmax_t) t);

    exit(EXIT_SUCCESS);

}

static time_t
```

```
my_mktime(struct tm *tp)
{
    int             e, isdst;
    time_t          t;
    struct tm       tm;
    unsigned char  wday[sizeof(tp->tm_wday)];
    e = errno;
    tm = *tp;
    isdst = tp->tm_isdst;
    memcpy(wday, &tp->tm_wday, sizeof(wday));
    tp->tm_wday = -1;
    t = mktime(tp);
    if (tp->tm_wday == -1) {
        memcpy(&tp->tm_wday, wday, sizeof(wday));
        return -1;
    }
    if (isdst == -1)
        tm.tm_isdst = tp->tm_isdst;
    if (   tm.tm_sec   != tp->tm_sec
        || tm.tm_min   != tp->tm_min
        || tm.tm_hour  != tp->tm_hour
        || tm.tm_mday  != tp->tm_mday
        || tm.tm_mon   != tp->tm_mon
        || tm.tm_year  != tp->tm_year
        || tm.tm_isdst != tp->tm_isdst)
    {
        errno = EINVAL;
        return t;
    }
    if (isdst != -1)
        goto out;
    tm = *tp;
    tm.tm_isdst = !tm.tm_isdst;
```

```
        tm.tm_wday = -1;
        mktime(&tm);
        if (tm.tm_wday == -1)
            goto out;
        if (tm.tm_isdst != tp->tm_isdst) {
            errno = ENOTUNIQ;
            return t;
        }
    out:
        errno = e;
        return t;
    }
```

SEE ALSO

date(1), gettimeofday(2), time(2), utime(2), clock(3), difftime(3), strftime(3), strp?
time(3), timegm(3), tzset(3), time(7)