Advanced Programming in the UNIX Environment

Week 04, Segment 8:
time(3) is an illusion

Department of Computer Science
Stevens Institute of Technology

Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/631/
#include <time.h>

time_t time(time_t *tloc);

Returns: time_t if OK, -1 on error

The time() function returns the value of time in seconds since 0 hours, 0 minutes, 0 seconds, January 1, 1970, Coordinated Universal Time.

(We already talked about the Y2K38 problem - see Week 01, Segment 3.)
```c
#include <err.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    time_t t;
    if ((t = time(NULL)) < 0) {
        err(EXIT_FAILURE, "unable to call time()");
        /* NOTREACHED */
    }
    printf("time(3) says: %ld\n", t);
    return EXIT_SUCCESS;
}
```

```
apue$ cc time1.c
apue$ ./a.out
output1
```
static char sccsid[] = "@(#)time.c 8.1 (Berkeley) 6/4/93";
#else
#endif
#endif /* LIBC_SCCS and not lint */

#include "namespace.h"
#include <sys/types.h>
#include <sys/time.h>

#include <time.h>

time_t
time(time_t *t)
{
    struct timeval tt;
    
    if (gettimeofday(&tt, NULL) == -1)
        return (time_t)-1;
    if (t != NULL)
        *t = (time_t)tt.tv_sec;
    return (time_t)tt.tv_sec;
gettimeofday(2)

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

int gettimeofday(struct timeval * restrict tp, void * restrict tzp);
```

Returns: 0 if OK, -1 on error

The system's notion of the current UTC time is obtained with the `gettimeofday()` call. The time is expressed in seconds and `microseconds` since midnight (0 hour), January 1, 1970.

```c
struct timeval {
    time_t       tv_sec;         /* seconds */
suseconds_t   tv_usec;        /* and microseconds */
};

struct timezone {
    int          tz_minuteswest; /* of Greenwich */
    int          tz_dsttime;     /* type of dst correction to apply */
};
```

Jan Schaumann
```c
int
main() {
    time_t t;
    struct timeval tv;

    if ((t = time(NULL)) < 0) {
        err(EXIT_FAILURE, "unable to call time()");
    }

    printf("time(3) says: %ld\n", t);

    if (gettimeofday(&tv, NULL) < 0) {
        if (gettimeofday(&tv, NULL) < 0) {
            err(EXIT_FAILURE, "unable to gettimeofday()"),
        /* NOTREACHED */
    }
    printf("gettimeofday(2) says: %ld.%d\n", tv.tv_sec, tv.tv_usec);

    return EXIT_SUCCESS;
}
```
#include <sys/time.h>

int clock_gettime(clockid_t clock_id, struct timespec *tp);

Returns: 0 if OK, -1 on error

The `clock_gettime()` function stores the time of the clock identified by `clock_id` into the location specified by `tp`; `clock_id` `CLOCK_REALTIME` represents present the amount of time (in seconds and `nanoseconds`) since 00:00 Universal Coordinated Time, January 1, 1970.

```c
apue$ diff -bu time[23].c
--- time2.c 2020-09-27 17:10:59.846633567 +0000
+++ time3.c 2020-09-27 17:11:48.040663199 +0000
@@ -7,6 +7,7 @@
    time_t t;
    struct timeval tv;
+
    struct timespec ts;

    if ((t = time(NULL)) < 0) {
        err(EXIT_FAILURE, "unable to call time()");
@@ -20,5 +21,11 @@
    }
    printf("gettimeofday(2) says: %ld.%ld\n", tv.tv_sec, tv.tv_usec);
+
    if (clock_gettime(CLOCK_REALTIME, &ts) < 0) {
        err(EXIT_FAILURE, "unable to call clock_gettime()");
        /* NOTREACHED */
+
    }
    printf("clock_gettime(2) says: %ld.%ld\n", ts.tv_sec, ts.tv_nsec);
+
    return EXIT_SUCCESS;
}
```

```
apue$ cc time3.c
apue$ ./a.out
  time(3) says: 1601230621
  gettimeofday(2) says: 1601230621.192601
  clock_gettime(2) says: 1601230621.192604618
apue$
```
Breaking time using `gmtime(3)`

```c
#include <time.h>

struct tm *gmtime(const time_t *clock);

Returns: pointer to struct_tm if OK, NULL on error
```

The `gmtime()` function converts to Coordinated Universal Time (UTC) and returns a pointer to the `tm` structure described in `tm(3)`.
gmtime(3) and the struct tm

struct tm {
    int     tm_sec;         /* seconds after the minute [0-61] */
    int     tm_min;         /* minutes after the hour [0-59] */
    int     tm_hour;        /* hours 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;       /* Daylight Savings Time flag */
};
CS631 - Advanced Programming in the UNIX Environment

INTERNATIONAL EARTH ROTATION AND REFERENCE SYSTEMS SERVICE (IERS)
SERVICE INTERNATIONAL DE LA ROTATION TERRESTRE ET DES SYSTEMES DE REFERENCE

SERVICE DE LA ROTATION TERRESTRE DE L’IERS
OBSERVATOIRE DE PARIS
61, Av. de l’Observatoire 75014 PARIS (France)
Tel. : +33 1 40 51 23 35
e-mail : services.iers@obspm.fr
http://hpliers.obspm.fr/eop-pc

Paris, 6 July 2016
Bulletin C 52

To authorities responsible for the measurement and distribution of time

UTC TIME STEP
on the 1st of January 2017

A positive leap second will be introduced at the end of December 2016.
The sequence of dates of the UTC second markers will be:

- 2016 December 31, 23h 59m 59s
- 2016 December 31, 23h 59m 60s
- 2017 January 1, 0h 0m 0s

The difference between UTC and the International Atomic Time TAI is:

- from 2015 July 1, 0h UTC, to 2017 January 1 0h UTC : UTC-TAI = -36s
- from 2017 January 1, 0h UTC, until further notice : UTC-TAI = -37s

No leap second will be introduced at the end of December 2020.
The difference between Coordinated Universal Time UTC and the International Atomic Time TAI is:

- from 2017 January 1, 0h UTC, until further notice : UTC-TAI = -37s

Leap seconds can be introduced in UTC at the end of the months of December or June, depending on the evolution of UT1-TAI.
Bulletin C is mailed every six months, either to announce a time step in UTC, or to confirm that there will be no time step at the next possible date.

Jan Schaumann
Unix Epoch and leap seconds

1483228800 should be 'Sat Dec 31 23:59:60 2016'; is: Sun Jan 1 00:00:00 2017

Sat Dec 31 23:59:58 2016 should be 1483228798; is 1483228798
Sat Dec 31 23:59:59 2016 should be 1483228799; is 1483228799
Sat Dec 31 23:59:60 2016 should be 1483228800; is 1483228800
Sun Jan 1 00:00:00 2017 should be 1483228801; is 1483228800

struct tm {
    int tm_sec; /* seconds after the minute [0-61] */
    ...
}

struct tm {
    int tm_sec; /* Seconds. [0-60] (1 leap second) */
    ...
}

Jan Schaumann
gmtime(3) and the struct tm

$ date +%s
1601230776
$ date
Sun Sep 27 18:19:37 UTC 2020

#include <time.h>

struct tm *gmtime(const time_t *clock);

Returns: pointer to struct_tm if OK, NULL on error

The gmtime() function converts to Coordinated Universal Time (UTC) and returns a pointer to the tm structure described in tm(3).
gmtime(3) and the struct tm

$ date +%s
1601230776
$ date
Sun Sep 27 18:19:37 UTC 2020

```
#include <time.h>

struct tm *gmtime(const time_t *clock);

Returns: pointer to struct_tm if OK, NULL on error

char *asctime(const struct tm *tm);

Returns: string in "Thu Nov 24 18:22:48 1986\n\0" format if OK, NULL on error
```
```
    time_t t;
    struct timeval tv;
    struct timespec ts;
    +struct tm *tm;

    if ((t = time(NULL)) < 0) {
        err(EXIT_FAILURE, "unable to call time()");
    }
    printf("clock_gettime(2) says: %ld.%ld\n", ts.tv_sec, ts.tv_nsec);
    +if ((tm = gmtime(&t)) == NULL) {
        +err(EXIT_FAILURE, "unable to call gmtime()");
        +/*@ NOTREACHED */
        +}
    +printf("asctime(3) says: %s", asctime(tm));
    +
    return EXIT_SUCCESS;
    }
```
gmtime(3) and the struct tm

struct tm {
    int    tm_sec;         /* seconds after the minute [0-61] */
    int    tm_min;         /* minutes after the hour [0-59] */
    int    tm_hour;        /* hours 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;       /* Daylight Savings Time flag */
};
gmtime(3) and the struct tm

```c
struct tm {
    int     tm_sec;         /* seconds after the minute [0-61] */
    int     tm_min;         /* minutes after the hour [0-59] */
    int     tm_hour;        /* hours 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;       /* Daylight Savings Time flag */
    long    tm_gmtoff;      /* offset from UTC in seconds */
    __aconst char *tm_zone; /* timezone abbreviation */
};
```
gmtime(3) and the struct tm

```
#include <time.h>

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

Returns: pointer to struct_tm if OK, NULL on error

char *asctime(const struct tm *tm);

Returns: string in "Thu Nov 24 18:22:48 1986\n\0" format if OK, NULL on error
```

$ date +%s
1601240163
$ TZ=US/Easter date
Sun Sep 27 20:56:10 GMT 2020
apue$ diff -bu time[45].c
--- time4.c   2020-09-27 20:03:22.934565666 +0000
+++ time5.c   2020-09-27 20:42:56.421829266 +0000
@@ -34,5 +34,11 @@
   printf("asctime(3) says:    %s", asctime(tm));
+
   if ((tm = localtime(&t)) == NULL) {
       err(EXIT_FAILURE, "unable to call gmtime()");
       /* NOTREACHED */
   }
   printf("localtime is:    %s", asctime(tm));
+
   return EXIT_SUCCESS;
}

apue$ cc time5.c
apue$ ./a.out

  time(3) says: 1601239518
  gettimeofday(2) says: 1601239518.556678
  clock_gettime(2) says: 1601239518.556681863
  asctime(3) says: Sun Sep 27 20:45:18 2020
  localtime is: Sun Sep 27 20:45:18 2020
apue$
```c
apue$ diff -bu time[45].c
--- time4.c 2020-09-27 20:03:22.934565666 +0000
+++ time5.c 2020-09-27 20:42:56.421829266 +0000
@@ -34,5 +34,11 @@
 }

 printf("asctime(3) says: %s", asctime(tm));
 +
 + if ((tm = localtime(&t)) == NULL) {
 +     err(EXIT_FAILURE, "unable to call gmtime()");
 +     /* NOTREACHED */
 + }
 +
 + printf("localtime is: %s", asctime(tm));
 +
 + return EXIT_SUCCESS;
 }

apue$ cc time5.c
apue$ ./a.out
time(3) says: 1601239518
gettimeofday(2) says: 1601239518.556678
clock_gettime(2) says: 1601239518.556681863
asctime(3) says: Sun Sep 27 20:45:18 2020
localtime is: Sun Sep 27 20:45:18 2020
```
mktime(3) and the struct tm

```
#include <time.h>

time_t mktime(struct tm *tm);

Returns: time_t if OK, -1 on error

char *ctime(const time_t *clock);

Returns: string like asctime(3) if OK, NULL on error
```

mktime(3) operates in the reverse direction from gmtime(3)/localtime(3).

ctime(3) is like asctime(3), but takes a time_t * instead of a struct tm *.
```c
++ printf("localtime is: %s", asctime(tm));
++
++ tm->tm_sec = 0;
++ tm->tm_min = 0;
++ tm->tm_hour = 0;
++ tm->tm_mday = 1;
++ tm->tm_mon = 0;
++ tm->tm_year = 70;
++ tm->tm_isdst = 0;
++
++ if ((t = mktime(tm)) < 0) {
++   err(EXIT_FAILURE, "unable to call mktime()");
++   /* NOTREACHED */
++ }
++
++ printf("epoch date is: %s", ctime(&t));
++
++ return EXIT_SUCCESS;
```

```
cc time6.c
cc ./a.out
time(3) says: 1601242425
gmtime(2) says: 1601242425.588696
clock_gettime(2) says: 1601242425.588699900
asctime(3) says: Sun Sep 27 21:33:45 2020
localtime is: Sun Sep 27 21:33:45 2020
epoch date is: Thu Jan  1 00:00:00 1970
```
gmtime(3) and the struct tm

#include <time.h>

ssize_t strftime(char * restrict buf, size_t maxsize,  
                 const char * restrict format, const struct tm * restrict timeptr);

Returns: number of characters placed into buffer

$ date
Sun Sep 27 21:47:30 UTC 2020
$ date +%D
09/27/20
$ date +%Y-%m-%dT%H:%M:%SZ
2020-09-27T21:47:47Z
#include <time.h>

+/* ISO 8601 format is "YYYY-MM-DDTHH:MM:SSZ" */
+define ISO8601_LENGTH 20 + 1 /* +1 for NULL */
+
+ int
+ main() {
+   time_t t;
@@ -10,6 +13,8 @@
+   struct timespec ts;
+   struct tm *tm;
+
+   char buf[ISO8601_LENGTH];
+   
+   if ((t = time(NULL)) < 0) {
+     err(EXIT_FAILURE, "unable to call time()");
+     /* NOTREACHED */
@@ -54,5 +59,11 @@
+   } 
+   printf("epoch date is: %s", ctime(&t));
+   
+   if (strftime(buf, ISO8601_LENGTH, "%Y-%m-%dT%H:%M:%SZ", tm) == 0) {
+     err(EXIT_FAILURE, "strftime failed");
+     /* NOTREACHED */
+   }
+   printf("ISO8601 formatted: %s\n", buf);
+   return EXIT_SUCCESS;
+}

apue$ ./a.out
time(3) says: 1601244204
gmtime(2) says: 1601244204.907320
clock_gettime(2) says: 1601244204.907323021
asctime(3) says: Sun Sep 27 22:03:24 2020
localtime is: Sun Sep 27 22:03:24 2020
epoch date is: Thu Jan 1 00:00:00 1970
ISO8601 formatted: 1970-01-01T00:00:00Z
apue$
Links and Discussions

- https://www.iana.org/time-zones
- https://en.wikipedia.org/wiki/Leap_second
- https://pubs.opengroup.org/onlinepubs/9699919799/xrat/V4_xbd_chap04.html#tag_21_04_16