Manejo de hora y fecha en C
`<time.h>`

Sebastián Santisi

2020-06-15

El lenguaje de programación C provee soporte en su biblioteca para el manejo de fechas y tiempo a través de dos tipos diferentes: `struct tm` y `time_t`, ambos contenidos en el encabezado `<time.h>`.

El tipo `struct tm` se utiliza para una representación de alto nivel accediendo a los campos de la fecha a través de los miembros de una estructura.

El tipo `time_t` es una representación compacta de una fecha en un tipo no transparente.

Mientras que el tipo `struct tm` se utiliza para manipular fechas, el tipo `time_t` se utiliza para almacenarlas, dado que representa la misma información en una variable numérica.

Representación del tiempo

Antes de entrar en el detalle de C es importante hacer un comentario sobre los sistemas informáticos y el tiempo.

Por lo general la mayor parte de los sistemas almacena las fechas como valores numéricos que representan los segundos transcurridos con respecto a determinada fecha de referencia llamada *epoch*. Por ejemplo en los sistemas de tipo Unix se cuentan los segundos transcurridos desde el 01/01/1970, en DOS los segundos desde el 01/01/1980, mientras que en los sistemas basados en Windows NT se cuenta la cantidad de 100 nanosegundos desde el 01/01/1601. Esto, entre otras cosas, limita las fechas mínimas y máximas que pueden representarse en un sistema. Por ejemplo, para sistemas Unix de 32 bits, el 19/01/2038 a las 03:14:07 se va a alcanzar la máxima fecha representable.

Dado que en el mundo conviven sistemas de todo tipo y que las fechas tienen que ser consistentes y comparables, sería ideal que los sistemas tuvieran su reloj interno en formato de fecha universal (UTC, Universal Time Coordinate, antes llamado GMT, Greenwich Mean Time). Esto fue así tradicionalmente hasta la incorporación de los sistemas de Microsoft que por limitaciones de software requieren que la hora del procesador esté en hora local.

Almacenar horas en formato local tiene múltiples inconvenientes, entre ellos que dos componentes que están en husos horarios diferentes no pueden compararse, o que cuando se aplican modificaciones de horario de verano pueden repetirse más de una vez las mismas horas. Se recomienda, independientemente de las limitaciones del sistema, representar los tiempos en UTC y realizar las conversiones pertinentes deste y hacia a hora local al almacenarlos y presentarlos.

El tipo `struct tm`

El tipo `struct tm` es una estructura con la siguiente definición:

```c
struct tm {
    // Hora:
    int tm_sec, // Segundos desde el minuto
```

```
    tm_min,     // Minutos desde la hora
    tm_hour,    // Horas desde la medianoche

    // Fecha:
    tm_mday,    // Día del mes
    tm_mon,     // Meses desde enero (Importante: el rango es 0 al 11)
    tm_year,    // Años desde 1900

    // Calendario:
    tm_wday,    // Días desde el domingo
    tm_yday,    // Días desde el 1 de enero

    tm_isdst;   // Flag booleano que indica si es horario de verano
};
```

Como se puede apreciar todos los miembros de la estructura son de tipo `int` por lo que no es un tipo diseñado para economizar memoria en lo más mínimo.

Hay dos maneras de tener un `struct tm`, una es llenando los miembros manualmente, la otra es a partir de un `time_t`.

Observaciones importantes:

- Para que un `struct tm` esté bien formado tienen que estar llenos todos sus miembros (con excepción de `tm_wday` y `tm_yday`), si no las funciones de biblioteca pueden fallar.
- Los valores de hora, minuto, segundo y día son intuituvos, ahora bien, prestar atención a que los meses se cuentan como la cantidad desde enero (ejemplo, diciembre valdrá 11) y que los años como cantidad desde el 1900 (ejemplo, el 2020 será 120).
- Se permite que los segundos valgan hasta 61, y la biblioteca hará la propagación al siguiente minuto.
- Si no se conoce el estado del horario de verano, dejarlo en `0`, es importante que este campo esté especificado.

Por ejemplo, si quisiéramos representar las 19:21:18 del día 15/06/2020 podríamos declarar:

```
struct tm fecha = {
    .tm_hour = 19,
    .tm_min = 21,
    .tm_sec = 18,

    .tm_mday = 15,
    .tm_mon = 5,
    .tm_year = 120,

    .tm_isdst = 0
};
```

El tipo `time_t`

El tipo `time_t` funciona como un tipo abstracto de datos, es decir, un tipo que manipulamos a través de las funciones de la biblioteca pero del que no se proveen detalles de cómo está implementado (y por lo tanto puede variar entre implementaciones).

Lo único que sabemos de `time_t` es que es un *tipo aritmético*. Esto significa que si tenemos tres variables `time_t t1, t2, t3;` podemos:

- **Compararlas:** `t1 < t2` va a ser verdadero si `t1` ocurre antes en el tiempo que `t2`.
- **Sustraerlas:** `(t2 - t1)` será la diferencia (en unidades de `time_t` que las desconocemos) entre los dos tiempos, y será comparable a otro intervalo `(t3 - t2)` dado que todo opera en la misma escala.
- **Adicionarlas:** `t3 + (t2 - t1)` le suma a `t3` el intervalo que separa a las variables `t1` y `t2`.

Salvo esas operaciones aritméticas básicas, el resto de las operaciones no serán transparentes.

Biblioteca

El encabezado `<time.h>` provee además funciones para interactuar con los tipos y para convertir entre `struct tm` y `time_t`.

La biblioteca es bastante antigua y se ha mantenido la interfaz por compatibilidad, con lo que hay varias cosas que pueden prestarse a confusión que vale la pena aclarar:

- Muchas de las funciones devuelven por el nombre y por la interfaz a la vez. En estas funciones si se le pasa `NULL` como parámetro sólo devuelven por el nombre.
- Muchas de las funciones devuelven punteros a cosas, pero no utilizan memoria dinámica, son punteros a variables estáticas.
- La biblioteca es anterior a que C permitiera la asignación de estructuras, por lo que siempre se trabaja con punteros (esto por otro lado la hace más eficiente).

Funciones de conversión

- `time_t mktime(struct tm *tp);` convierte a `tp` en `time_t`, si no puede devuelve `-1` (validar siempre esto).
- `struct tm *gmtime(const time_t *tp);` convierte a `tp` en `struct tm` en UTC, devuelve `NULL` si el sistema no lo soporta.
- `struct tm *localtime(const time_t *tp);` convierte a `tp` en `struct tm` en hora local.

Hora actual

- `time_t time(time_t *tp);` devuelve la fecha y hora actual o `-1` si no puede obtenerse este dato.

Intervalos

- `double difftime(time_t t1, time_t t2);` devuelve la diferencia **en segundos** entre `t2` y `t1`.

Conversión a cadena

La biblioteca ofrece funciones de alto nivel para generar cadenas con formato

- `char *asctime(const struct tm *tp);` devuelve una cadena con un formato `"Sun Jan 3 13:08:42 1988\n"`.
- `char *ctime(const time_t *tp);` equivalente a `asctime(localtime(tp));`.
- `size_t strftime(char *s, size_t smax, const char *fmt, const struct tm *tp);` imprime a `tp` en la cadena `s` de hasta `smax` caracteres, acorde con el formato `fmt`. Devuelve la cantidad de caracteres escritos.

  El formato es análogo al de las funciones de la familia de `printf()` pero los modificadores `%` en este caso sirven para formatear fechas.

Formato	Significado
`%A`	Día de la semana
`%a`	Día de la semana abreviado
`%B`	Nombre del mes
`%b`	Nombre del mes abreviado

Formato	Significado
`%c`	Representación local de día y hora
`%d`	Día del mes [00-31]
`%H`	Hora (24h) [00-23]
`%I`	Hora (12h) [01-12]
`%j`	Día del año [001-366]
`%M`	Minutos [00-59]
`%m`	Mes [01-12]
`%p`	Indicador de `"AM"` o `"PM"`
`%S`	Segundos [00-61]
`%U`	Semana del año (empezando en domingo) [00-53]
`%W`	Semana del año (empezando en lunes) [00-53]
`%w`	Día numérico de la semana (desde domingo) [0-6]
`%X`	Representación local de tiempo
`%x`	Representación local de día
`%Y`	Año completo con el siglo
`%y`	Año sin el siglo [00-99]
`%Z`	Nombre de la zona horaria
`%%`	Imprime un `'%'`

Por ejemplo, puede generarse la cadena que genera `asctime(fecha);` con

```
char s[MAX_STR];
strftime(s, MAX_STR, "%a %b %d %H:%M:%S %Y\n", fecha);
```

Manejo de tiempos internos

Cabe destacar que los tipos y funciones presentados hasta el momento sirven para el manejo de fechas general y no para tomar tiempos internos de funcionamiento de un programa.

Para esos usos se provee un tipo adicional `clock_t` que es un tipo aritmético para tiempos de procesador.

El mismo cuenta la cantidad de *clocks* desde el inicio de la aplicación y se provee la macro `CLOCKS_PER_SEC` para saber cuántos clocks representan un segundo.

La función `clock_t clock(void);` devuelve el tiempo de procesador actual o `-1` si no está disponible.

Para medir el tiempo de ejecución de un bloque se debe almacenar el tiempo de procesador antes de iniciar y compararlo con el de después de terminar.