Midiendo e imprimiendo el tiempo en Unix

Dr. Roberto Gómez Cárdenas

12 de marzo de 1999

Resumen

El presente documento esta dirigido a desarrolladores de programas en C sobre plataforma Unix. Algunos de mis alumnnos tienen la duda de como imprimir las diferentes estructuras que definen la fecha y la hora en Unix, así como la forma de medir el tiempo de ejecución de dos programas para ver cual es más rápido. El presente papel tiende a aclarar dichas dudas. Para entender el contenido de este documento el lector debe contar con un buen manejo del lenguaje C.

1 Introducción

La medición del tiempo ha evolucionado bastante desde el reloj de arena, pero pasemos por alto algunos siglos y situemonos en 1948 cuando se inventó el reloj atómico. Para poder calcular la hora este reloj cuenta las transiciones del átomo del cesium 133 (un segundo es el tiempo para que el átomo 133 realize 9,192,631,770 transiciones). Existen aproximandamente 50 laboratorios en el mundo que cuentan con un reloj atómico que, periodicamente, informan su hora calculada al BIH (*Bureau International de l'Heure*-Oficina Internacional de la Hora) en París. El BIH promedia lo recibido y produce el TAI (*Temps Atomic Internationale*-Tiempo Atómico Internacional).

El BIH ajusta sus cálculos obtenidos para evitar desfazamientos en el tiempo. Esto da como resultado un sistema de tiempo basado en los segundos TAI conocido como UTC (Universal Coordinated Time). Este UTC es la base del tiempo civil y vino a reemplazar el antiguo estándar de tiempo GMT (Greenwich Mean Time).

Dentro de la computadora se pasa de otra forma. Existen dos tipos de relojes usados dentro de una computadora los cuales son diferentes de los utilizados por los seres humanos. Los más simples estan atados a una línea de 110 ó 220 volts, y provocan una interrupción en cada ciclo de voltaje, funcionando a 50 o 60 Hz.

El otro tipo de reloj es una combinación de tres componentes: un cristal oscilador (cuarzo) un contador y un registro. Cuando a la pieza de cristal se le somete un voltaje, este comienza a emitir periodicamente una señal. Dicha señal es utilizada para inicializar el contador y actualizar los registros.

Para prevenir que el tiempo se pierda cuando el sistema se apaga algunas computadoras cuentan con un reloj de respaldo que funciona en base a baterias. Si no se cuenta con este dispositivo, al arrancar el sistema este puede preguntar fecha y hora al usuario, o puede obtener el tiempo de un host remoto. Este último lo obtiene de alguna estación o servidor de tiempo ATI o UTC, el cual es calculado como se explicó anteriormete.

Lo que hace el sistema operativo Unix para calcular la fecha y hora es mantener un contador del número de segundos que han transcurrido desde las 00:00:00 UTC del 1o de enero de 1970 hasta el momento en que se solicitó la hora. Cada vez que se le solicita la hora y fecha al sistema, regresa este número.

Este documento se encuentra dividido en dos secciones, la primera de ellas presenta las principales funciones para imprimir la fecha y hora del sistema; la segunda presenta los códigos para poder medir el tiempo de ejecución de un programa y/o proceso. Al final se incluye una pequeña bibliografía de donde se obtuvó una buena parte del material de este documento. Sin embargo es bueno remarcar que mucha de esta información se obtuvó del comando `man` de Unix.

2 Llamadas de sistema para medir el tiempo

El elemento base de todas las funciones relacionadas con el cálculo de tiempo en Unix es un número que indica el número de segundos transcurridos desde las 00:00:00 del 1o. de enero de 970 hasta el momento actual. La funcón `time` permite obtener este dato y su sintaxis es.

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

time_t time(time_t *tloc);
```

La función regresa lo que se conoce como el *tiempo calendario* (segundos entre las 00:00:00 del 1/1/70 hasta el momento en que es invocada la función).

Generalmente el argumento `tloc` toma el valor de NULL. Si el argumento es diferente de NULL el valor de regreso también es almacenado en la localidad donde `tloc` apunte. En caso de error la función regresa un valor de -1.

Un ejemplo de uso se presenta en el código siguiente:

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

main()
{

    time_t hora;

    hora = time(NULL);

    printf("Este programa termino de ejecutarse a las %d \n", hora);
}
```

La compilación de este programa no requiere ninguna opción en particular. Dicha compilación y un ejemplo de ejecución se presentan a continuación:

```
rogomez@armagnac:978>cc ejemtime.c -o ejemtime
rogomez@armagnac:979>ejemtime
Este programa termino de ejecutarse a las 920003680
rogomez@armagnac:980>
```

Como se puede constatar la salida del programa no es muy útil, con la salida anterior el usuario estaría obligado a realizar cierto tipo de cálculo para poder saber la hora y fecha exacta. Unix proporciona la función `localtime()` que realiza dicho cálculo. La sintaxis de esa llamada es:

```
#include <time.h>

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

La función toma un tiempo calendario, realiza ciertos cálculos y llena la estructura `tm` con sus respectivos valores. El usuario puede imprimir los campos que más le interesen. Los campos de la estructura `struct tm` son:

`int tm_sec`	los segundos [0, 61]
`int tm_min`	minutos después de la hora [0, 59]
`int tm_hour`	horas después de media noche [0, 23]
`int tm_mday`	día del mes [1, 31]
`int tm_mon`	meses desde enero [0, 11]
`int tm_year`	años desde 1900

`int tm_wday` día de la semana, a partir del domingo [0, 6]

`int tm_yday` día del año, a partir del 1o. enero del año en curso [0, 365]

`int tm_isdst` bandera para alternar los ajustes de tiempo por cambio de horario

De esta forma es posible saber la fecha y hora de un determinado evento. El siguiente programa es una modificación del presentado anteriormente, el cual muestra la forma de manejar los diferentes campos de la estructura `tm`.

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

main()
{

   time_t hora;
   struct tm *tmp;
   static char *const nom_dia[] = {
      "domingo", "lunes", "martes", "miercoles",
      "jueves",  "viernes", "sabado", "desconocido"
   };

   static char *const nom_mes[] = {
      "enero", "febrero", "marzo",     "abril",  "mayo",     "junio",
      "julio", "agosto",  "septiembre","octubre","noviembre","diciembre"
   };

   hora = time(NULL);

   printf("Han pasado %d segundos desde las 00:00:00 del 1o. enero 1970 \n", hora);

   tmp =  (struct tm *)localtime(&hora);

   printf("Es decir son las %d hrs %d mns %d sgs.\n",tmp->tm_hour, tmp->tm_min, tmp->tm_sec);
   printf("del dia %d del mes %d (%s) de %d \n",tmp->tm_mday, tmp->tm_mon+1,
           nom_mes[tmp->tm_mon], tmp->tm_year+1900);
   printf("Es el %d dia de la semana (%s) y el %d dia del a~o \n",tmp->tm_wday,
           nom_dia[tmp->tm_wday],tmp->tm_yday);

}
```

Es importante comentar algunos detalles del código presentado; en primer lugar hay que sumarle 1900 al año (`tmp->tm_year`) a la hora de imprimirlo y sumarle un 1 al mes (`tmp->tm_mon`). También se puede apreciar que se utilizó un par de variables estáticas (`nom_dia` y `nom_mes`) para poder imprimir el día y el mes.

Al igual que el código anterior este no requiere de ninguna opción en su compilación ni en su ejecución:

```
rogomez@armagnac:126>cc imptime.c -o imptime
rogomez@armagnac:127>imptime
Han pasado 920941719 segundos desde las 00:00:00 del 1o. enero 1970
Es decir son las 19 hrs 8 mns 39 sgs.
del dia 8 del mes 3 (marzo) de 1999
Es el 1 dia de la semana (lunes) y el 66 dia del a~o
rogomez@armagnac:128>
```

La salida del programa es más compresible y es posible imprimir solo los datos, en el formato deseado, que el usuario necesite.

Aparte de las funciones anteriores se cuenta con las siguientes funciones, todas definidas dentro de `<time.h>`, para el manejo del tiempo.

```
struct tm *gmtime(const time_t *clock)
char *tm *asctime(const struct tm *tm )
char *ctime(const time_t *clock);
time_t mktime(struct tm *timeptr)
size_t strftime(char *buf, size_t maxsize, const char *format, const struct tm *tmptr)
```

La primera de ellas, `gmtime()`, es muy parecida a la función `localtime()`, con la única diferencia de que la función no toma en cuenta la zona de tiempo local manejada por el sistema y calcula directamente el tiempo UTC (Coordinated Universal Time), que es lo usado por el sistema Unix internamente.

La función `asctime()` convierte una estructura `tm` de tipo `struct tm` a una cadena de 26 caracteres, que despliega la fecha y hora en inglés respetando el siguiente formato:

```
Mon Mar  8 20:00:46 1999
```

La función `ctime()` realiza lo mismo que `asctime()`, la diferencia esta en el parámetro de entrada. Mientras que la primera tiene como parámetro una estructura `tm`, esta función recibe como parámetro una variable del tipo `time_t`. La salida tiene el mismo formato de salida para ambas.

La función `mktime()` convierte el tiempo representado por la estructura apuntado por el parámetro `timeptr` en un formato tiempo calendario. Aparte de calcular el tiempo de calendario (segundos) la función completa la estructura `tm`. Los valores originales de los campos `tm_wday` y `tm_yday` son ignorados, y los valores originales de los otros campos no estan restringidos a los rangos establecidos en la definición de la estructura. Si todo pasa bien los campos `tm_wday` y `tm_wyday` tendrán asignados sus valores correctos, y los valores de los otros campos serán tales que representen el tiempo calendario especificado. El valor final de `tm_mday` no es asignado hasta que el valor de `tm_mday` y `tm_year` es determinado.

El `tm_year` debe ser igual o superior a 1970. Valores anteriores a las 00:00:00 UTC del 1o. de enero de 1970 no pueden ser representados en tiempo calendario. Si el tiempo calendario no puede ser representado la función regresa un valor de -1.

Un ejemplo de uso, calcular el día de la semana que corresponde al 4 de agosto de 1975, se presenta en el siguiente código:

```
/* imprimiendo el dia de la semana del 14 octubre 1975 */

#include <stdio.h>
#include <time.h>

static char *const dia_semana[] = {
    "domingo", "lunes", "martes", "miercoles",
    "jueves",  "viernes", "sabado", "desconocido"}
};
struct tm tiempo;
/* ......  */
tiempo.tm_year  = 1975 - 1900;
tiempo.tm_mon   = 10 - 1;
tiempo.tm_mday  = 4;
tiempo.tm_hour  = 0;
tiempo.tm_min   = 0;
tiempo.tm_sec   = 1;
tiempo.tm_isdst = -1;
if (mktime( &tiempo) == -1)
   tiempo.tm_wday = 7;
printf("%s\n",dia_semana[tiempo.tm_wday]);
```

La más complicada de todas es la última función `strftime()`. Dicha función es muy parecida a la función `printf()` de C, y sirve para darle un formato de salida a los diferentes parámetros que definen el tiempo.

El argumento final `struct tm *tmptr` contiene los diferentes parámetros del tiempo a formatear. El resultado formateado se almacena en el arreglo de caracteres apuntado por `buf` de tamaño `maxsize`. Si la longitud del resultado de la función, incluyendo el carácter nulo, cabe en el arreglo la función regresa el número de caracteres almacenados en `buf` (excluyendo el carácter de terminación). En otro caso la función regresa un 0.

El parámetro `format` controla el formateo del valor del tiempo. Al igual que para la función `printf()`, se cuentan con especificadores de conversión que consisten de un carácter de porcentaje (%) precedidos de una letra. Todo el resto de los caracteres del formato son copiados en el parámtero de salida. Dos porcentajes seguidos (%%) generan un solo porcentaje. A diferencia del `printf()` cada conversión debe realizarse sobre un string de tamaño fijo.

La tabla de abajo presenta algunos de los especificadores de conversión de ANSI C.

Formato	**Descripción**	**Ejemplo**
%a	nombre abreviado del día de la semana	`Tue`
%A	nombre completo del día de la semana	`Tuesday`
%b	nombre abreviado del mes	`Jan`
%B	nombre completo del mes	`January`
%c	fecha y hora	`Tue Jan 14 19:40:30 1992`
%d	día del mes [01,31]	`14`
%H	hora en formato 24 hrs: [00,24]	`19`
%I	hora en formato 12 horas: [01,12]	`07`
%j	día del año [001, 366]	`014`
%m	mes:[01,12]	`01`
%M	minutos: [00,59]	`40`
%p	AM o PM	`PM`
%S	segundos: [00, 61]	`40`
%U	número semana de domingo: [00, 53]	`02`
%w	número día semana: [0=domingo, ... 6]	`2`
%W	número semana de lunes: [00,53]	`02`
%x	fecha	`01/14/92`
%X	hora	`19:40:30`
%y	año dentro del siglo: [00,99]	`92`
%Y	año con el siglo	`1992`
%Z	nombre tiempo zona	`MST`

La tercera columna de la tabla es la salida de `strftime` bajo el sistema SVR4. Los únicos dos formatos que no son comprensibles a primera vista son el `%U` y el `%W`. El primero es el número de semana del año que contiene el primer domingo. El formato `%W` es el número de semana del año de la semana que contiene el primer lunes.

Las funciones `localtime()`, `mktime()`, `ctime()` y `strftime()` estan afectadas por la variable de ambiente `TZ`, que almacena información acerca de la zona de tiempo. Si la variable esta definida como un string nulo, (e.g. `TZ=`) entonces la zona UTC es usada.

Un ejemplo de uso de las funciones anteriores se presenta a continuación:

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

main()
{

   time_t hora, segundos;
   struct tm *tmp1, *tmp2, tmp3;
   char *f1, *f2, *f3, f4[10], f5[50];
   static char *const dia[] = {
       "domingo", "lunes", "martes", "miercoles",
       "jueves",  "viernes", "sabado", "desconocido"
   };
```

```
        hora = time(NULL);

        printf("Han pasado %d segundos desde las 00:00:00 del 1/1/70\n",hora);

        tmp1 = (struct tm *)localtime(&hora);
        f1 = (char *)asctime(tmp1);
        printf("Regreso  localtime(%d)  y  asctime(): %s",hora,f1);

        tmp2 = (struct tm *)gmtime(&hora);
        f2 = (char *)asctime(tmp2);
        printf("Reg. gmtime(%d) y de asctime() (UTC): %s",hora,f2);

        f3 = ctime(&hora);
        printf("Regreso  de  la  llamada   ctime(%d): %s\n",hora,f3);

        if (strftime(f4, sizeof(f4), "%d", tmp1) != 0 )
           printf("Salida strftime() dia del mes: %s\n",f4);
        if (strftime(f5, sizeof(f5), "Nombre zona tiempo usada: %Z \n", tmp1) != 0 )
           printf("%s",f5);

        tmp3.tm_hour=11;
        tmp3.tm_min=23;
        tmp3.tm_sec=42;

        tmp3.tm_mday=26;
        tmp3.tm_mon=11-1;
        tmp3.tm_year=1973-1900;

        segundos=mktime(&tmp3);

        printf("Entre las 00:00:00 del 1/1/70 y las %d:%d:%d del %d/%d/%d \n",tmp3.tm_hour,
                tmp3.tm_min, tmp3.tm_sec, tmp3.tm_wday, tmp3.tm_mon+1, tmp3.tm_year);
        printf("han pasado %d segundos \n",segundos);
        printf("El %d/%d/%d fue un %s\n", tmp3.tm_wday, tmp3.tm_mon+1, tmp3.tm_year,
                dia[tmp3.tm_wday]);

}
```

La salida del código anterior se presenta a continuación. Como puede constatarse el programa se compiló sin necesidad de ligarlo a ninguna librería o biblioteca.

```
rogomez@armagnac:142>cc otrostime.c -o otrostime
rogomez@armagnac:143>otrostime
Han pasado 920944846 segundos desde las 00:00:00 del 1/1/70
Regreso  localtime(920944846)  y  asctime(): Mon Mar  8 20:00:46 1999
Reg. gmtime(920944846) y de asctime() (UTC): Tue Mar  9 02:00:46 1999
Regreso  de  la  llamada   ctime(920944846): Mon Mar  8 20:00:46 1999
Salida strftime() dia del mes: 08
Nombre zona tiempo usada: CST
Entre las 00:00:00 del 1/1/70 y las 11:23:42 del 1/11/73
han pasado 123182622 segundos
El 1/11/73 fue un lunes
rogomez@armagnac:144>
```

Para ejemplificar la salida de la función `mktime()` se decidió calcular y desplegar el día de la semana del 26 de noviembre de 1973, mientras que para la función `strftime()` se imprimió la zona de tiempo usada y el día del mes.

3 Midiendo el tiempo de ejecución de un programa

Una forma sencilla de medir el tiempo de ejecución de un programa es obtner y registrar la hora al principio del programa y al final de este. Una vez el cálculo terminado se calcula la diferencia entre las dos horas y se despliega el resultado.

En vista de que los segundos de la función `time()` estan calculados a partir de la misma fecha, no no es necesario darle un formato a la lectura del tiempo realizada con dicha función. La solución, aparentemente, es simple; basta con llamar dicha función al principio y al final del programa y calcular la diferencia de los valores regresados. El código de esto se presenta a continuación:

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

main(int argc, char *argv[])
{

    time_t hora_inic, hora_fin;
    int n,i,temps;

    n = atoi(argv[1]);

    hora_inic = time(NULL);
    printf("Empieza ejecucion a las %d \n",hora_inic);

    printf("durmiendo ... \n");
    usleep(n);

    hora_fin = time(NULL);
    printf("Termina ejecucion a las %d \n",hora_fin);

    temps=hora_fin - hora_inic;
    printf("Tiempo ejecucion: %d segs\n",temps);
}
```

El programa no realiza ningún cálculo en especial, tan sólo lee un número de la línea de comandos y se duerme durante ese tiempo. La función `usleep()` provoca que el proceso se duerma durante n microsegundos, esta función puede substituirse por un código que realize cierto cálculo. Algunas salidas del programa se presentan a continuación:

```
rogomez@armagnac:166>cc mide1.c -o mide1
rogomez@armagnac:167>mide1 7000000
Empieza ejecucion a las 920948377
durmiendo ...
Termina ejecucion a las 920948384
Tiempo ejecucion: 7 segs
rogomez@armagnac:168> mide1 3
Empieza ejecucion a las 920948176
durmiendo ...
Termina ejecucion a las 920948176
Tiempo ejecucion: 0 segs
rogomez@armagnac:169>mide1 5000
Empieza ejecucion a las 920948189
durmiendo ...
Termina ejecucion a las 920948189
Tiempo ejecucion: 0 segs
rogomez@armagnac:170>
```

Como puede observarse la primera ejecución tomo siete segundos ($7000000X10^{-6} = 7$). Sin embargo las dos últimas ejecuciones producieron un tiempo de ejecución igual, 0 segundos; cuando el proceso estuvo "en ejecución" 3 y 5000 microsegundos respectivamente. El problema de utilizar

la función `time()` es que esta trabaja en segundos, por lo que ejecuciones inferiores a un segundo no es posible "medirlas" con esta llamada.

Sin embargo, Unix cuenta con otra función para obtener la hora, `gettimeofday()`. La sintaxis de esta llamada es:

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

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

En realidad esta función obtiene la noción del sistema del tiempo actual. En este caso el tiempo actual es expresado en segundos y *microsegundos* transcurridos desde las 00:00 del UTC del 1o. de enero de 1970. La resolución depende del sistema de reloj que utilice el hardware; el tiempo puede ser actualizado continuamente o en ticks de reloj.

El argumento `tp` apunta a una estructura de tipo `struct timeval`, la cual incluye los siguientes campos:

- `long tv_sec;` segundos desde el 1/1/1970
- `long tv_usec;` microsegundos desde el 1/1/1970

Si el parámetro `tp` es un apuntador a NULL la información no es regresada. El segundo argumento debe ser un apuntador a NUUL.

Como dato aparte, es bueno saber que existe una función llamada `settimeofday()` con los mismos parámetros que sirve para establecer el tiempo actual del sistema. Sin embargo solo el superusuario tiene los permisos necesarios para utilizar esta función.

Entonces, para medir tiempos de ejecución menores a un segundo podemos utilizar dicha función en combinación con el principio utilizado anteriormente. La función `gettimeofday` será llamada al principio y final del programa, y el tiempo de ejecución será la diferencia de lo regresado por estas dos llamadas.

El programa anterior modificado da como resultado el siguiente código:

```
rogomez@armagnac:187>more mide2.c
#include <sys/types.h>
#include <sys/time.h>

main(int argc, char *argv[])
{

   struct timeval inic, fin, temps;
   int n;

   n = atoi(argv[1]);

   /* hora al iniciar el proceso de calculo */
   gettimeofday(&inic,NULL);
   printf("Empieza ejecucion a las %d:%d \n",inic.tv_sec, inic.tv_usec);

   /* proceso de calculo a medir */
   printf("durmiendo ... \n");
   usleep(n);

   /* hora de termino del proceso de calculo */
   gettimeofday(&fin,NULL);
   printf("Termina ejecucion a las %d:%d \n",fin.tv_sec, fin.tv_usec);

   /* calculo del tiempo transcurrido */
   temps.tv_sec  = fin.tv_sec - inic.tv_sec;
   temps.tv_usec = fin.tv_usec - inic.tv_usec;

   printf("Tiempo ejecucion: %d segs y %d usgs \n",temps.tv_sec, temps.tv_usec);
}
```

El código es el mismo que el presentado al principio de esta sección. La parte de ejecución a medir es el tiempo que se va a dormir el proceso (en microsegundos). La salida de tres ejecuciones se presenta a continuación:

```
rogomez@armagnac:188>cc mide2.c -o mide2
rogomez@armagnac:189>mide2 3
Empieza ejecucion a las 920949210:594460
durmiendo ...
Termina ejecucion a las 920949210:606869
Tiempo ejecucion: 0 segs y 12409 usgs
rogomez@armagnac:190>mide2 5000
Empieza ejecucion a las 920949236:209988
durmiendo ...
Termina ejecucion a las 920949236:227388
Tiempo ejecucion: 0 segs y 17400 usgs
rogomez@armagnac:191>mide2 7000000
Empieza ejecucion a las 920949268:427833
durmiendo ...
Termina ejecucion a las 920949275:438497
Tiempo ejecucion: 7 segs y 10664 usgs
rogomez@armagnac:192>
```

,

En este caso las dos primeras ejecuciones fueron menores a un segundo y su tiempos de ejecución fueron de 12409 usgs y de 17400 usgs respectivamente. En el caso de la última el tiempo fue de 7 sgs y 10664 usgs. Como se puede ver, este metodo nos ofrece un poco más de exactitud que el anterior pero presenta un pequeño detalle.

Si observamos la salida de la última ejecución, se despliega un tiempo de ejecución de 7 sgs y 10664 usgs, y se le pidió al proceso que se fuera a dormir 7segs, por lo que sobran 10664 usgs. Lo mismo puede apreciarse con el resto de las salidas.

Dos detalles de la función pueden explicar lo anterior; la resolución del campo que mide los microsegundos (`tv_usec`) depende de la frecuencia del reloj del sistema, que puede ser bastante pequeña en comparación con la del reloj de la computadora. Otra detalle es que el programa depende de la carga del sistema. Si el programa no tiene otros procesos compitiendo por la CPU, tardará menos en ejecutarse que si tiene otros competidores.

En la siguiente sección se verá como se puede medir el tiempo real de ejecución CPU de un proceso.

3.1 Tiempo ejecución real, ejecución sistema y de usuario

La llamada de sistema `times()` reporta los tiempos de procesamiento del proceso padre y del hijo. Dicha función cuenta con la siguiente sintaxis:

```
#include <sys/times.h>
#include <limits.h>

clock_t times(struct tms *buffer);
```

Esta función llena la estructura `tms` apuntada por el parámetro `buffer` con información de *tiempo-de-auditoria*. La estructura `tms` esta definida en `<sys/times.h>` y esta formada por los siguientes campos:

- `clock_t tms_utime;` tiempo CPU utilizado mientras se ejecutan las instrucciones en el espacio del usuario del proceso invocador de la función
- `clock_t tms_stime;` tiempo CPU utilizado por el proceso en modo kernel
- `clock_t tms_cutime;` es la suma del `tms_utime` y del tiempo `tms_utime` de los procesos hijos

- `clock_t tms_cstime;` es la suma del `tms_stime` y del tiempo `tms_cstime` de los procesos hijos

Todos los tiempos se encuentran expresados en clicks de reloj. El valor especifico de un click de reloj se encuentra definido en la variable `CLK_TCK`, que se encuentra en el archivo `<limits.h>`. Es posible leer el valor de esta variable a través de la llamda de sistema `sysconf()`.

Los tiempos de ejecución de un proceso hijo se incluyen en en los campos `tms_cutime` y `tms_cstime` del padre, cuando las llamadas `wait()` y `waitpid()` regresan el identificador del proceso hijo que terminó. Si un proceso no espero por sus hijos sus tiempos no serán incluidos en sus tiempos.

El cálculo de todos los tiempos anteriores no toma en cuenta el tiempo dedicado por los procesos del sistema a los procesos del usuario (p.e. tiempo cambio de contexto de un proceso). Los tiempos anteriores son tiempos reales de CPU, por lo que no se contabilizan los períodos en los que el proceso duerme (uso de la llamada `sleep()`).

Si la función `times` se ejecuta satisfactoriamente, esta devuelve el tiempo real transcurrido, en tics de reloj, contando a partir de un instante pasado arbitrario. Este puede ser el momento de arranque del sistema y no cambia de una llamada a otra. Si la función falla, esta regresa un valor de -1.

El código presentado a continuación nos ejemplifica el uso de `times` para medir el tiempo de ejecución.

```
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include <sys/times.h>

main()
{

   FILE *fd;
   struct tms inic, fin;
   clock_t t1, t2;
   long clicks;
   int n,i;
   float res;

   clicks=sysconf(_SC_CLK_TCK);
   printf("Se dan %d tics por segundo \n",clicks);

   /* tomando tiempo antes del calculo*/
   t1 =times(&inic);

   /* calculo a realizar */
   for(i=0; i<100; i++) {
     n=(int)(n*i/(i+1293));
     n=(n*n*n*n*n)/(2);
     res=sqrt(25000);
     fd=fopen("toto","w");
     fprintf(fd,"Esto es una prueba\n");
     fclose(fd);
   }
   /* fin del calculo */

   /* tomando tiempo despues calculo */
   t2 =times(&fin);

   /* desplegando resultados */
   printf("Tiempos de ejecucion: \n");
   printf("\tReal: %d clicks (%f sgs) \n", t2-t1, (float)((float)(t2-t1) / (float)clicks));
```

```
    res = (float)( ( (float)fin.tms_utime-(float)inic.tms_utime) / (float)clicks);
    printf("\tEn modo usuario: %d clicks (%f sgs) \n",fin.tms_cutime-inic.tms_cutime,res);
    res = (float)( ( (float)fin.tms_stime-(float)inic.tms_stime) / (float)clicks);
    printf("\tEn modo kernel: %d clicks (%f sgs) \n",fin.tms_stime-inic.tms_stime,res);
}
```

El cálculo consiste en hacer 100 veces unos calculos al azar y en abrir un archivo llamado *toto*, escribir "Esto es una prueba" en él y cerrarlo (también 100 veces). Un ejemplo de compilación y de ejecución del código anterior se presenta a continuación:

```
rogomez@armagnac:289>cc mide3.c -lm -o mide3
rogomez@armagnac:290>mide3
Se dan 100 tics por segundo
Tiempos de ejecucion:
        Real: 133 clicks (1.330000 sgs)
        En modo usuario: 0 clicks (0.000000 sgs)
        En modo kernel: 1 clicks (0.010000 sgs)
rogomez@armagnac:291>mide3
Se dan 100 tics por segundo
Tiempos de ejecucion:
        Real: 178 clicks (1.780000 sgs)
        En modo usuario: 0 clicks (0.010000 sgs)
        En modo kernel: 0 clicks (0.000000 sgs)
rogomez@armagnac:292>mide3
Se dan 100 tics por segundo
Tiempos de ejecucion:
        Real: 132 clicks (1.320000 sgs)
        En modo usuario: 0 clicks (0.000000 sgs)
        En modo kernel: 0 clicks (0.000000 sgs)
rogomez@armagnac:293>mide3
Se dan 100 tics por segundo
Tiempos de ejecucion:
        Real: 132 clicks (1.320000 sgs)
        En modo usuario: 0 clicks (0.030000 sgs)
        En modo kernel: 25 clicks (0.250000 sgs)
rogomez@armagnac:294>
```

Como puede apreciarse, aún hay algunas discrepancias en los tiempos de ejecución (1.33, 1.78, 1.32, 1.32), pero estas son mínimas en comparación con las salidas de los mediciones realizadas en la sección anterior con otras funciones.

Otro ejemplo de uso de la llamada `times` se presenta en forma de la implementación del comando `time` de Unix. Dicho comando invoca un comando o programa con todo y argumentos, y escribe un mensaje que lista tiempos de ejcución del comando o programa. El código es el presentado en [2].

```
    /*  Uso: tiempo <proceso-a-ejecutar>  */

#include <stdio.h>
#include <fcntl.h>
#include <time.h>
#include <sys/times.h>
#include <signal.h>
#include <errno.h>
#include <unistd.h>

main(argc, argv)
    int    argc;
    char  *argv[];
{

    struct tms bt1, bt2;
```

```
    clock_t t1, t2;
    int pid, w, estado, tty;

    /* verificando parametros */
    if (argc < 2) {
       fprintf(stderr,"Error uso: %s <comando> \n", argv[0]);
       exit(-1);
    }

    /* Apertura del terminal asociado al proceso */
    if ( (tty = open("/dev/tty", O_RDWR)) == -1) {
       perror(argv[0]);
       exit(-1);
    }

    /* desplegando numero de clikcs */
    clicks=sysconf(_SC_CLK_TCK);
    printf("Se dan %d tics por segundo \n",clicks);

    t1 = times(&bt1);

   /* Creacion procesos padre e hijo */
   if ((pid = fork()) == -1) {
     perror(argv[0]);
     exit(-1);
   }
   else
      if (pid == 0) {
        close(0); dup(tty);
        close(1); dup(tty);
        close(2); dup(tty);

        execvp(argv[1], &argv[1]);
        perror("execvp");
        exit(127);
     }
     else  {
        close(tty);
        signal(SIGINT, SIG_IGN);
        signal(SIGQUIT, SIG_IGN);
        while ( (w = wait(&estado)) != pid && w!= -1);
        t2 = times(&bt2);
        signal(SIGINT, SIG_DFL);
        signal(SIGQUIT, SIG_DFL);

        /* Presentacion resultados */
        printf("Tiempos de ejecucion: \n");
        printf("\tReal: %g sgs. \n", (float)(t2-t1) / _SC_CLK_TCK);
        printf("\tEn modo usuario: %g sgs. \n",
                  (float)(bt2.tms_cutime - bt1.tms_cutime) / _SC_CLK_TCK);
        printf("\tEn modo kernel: %g sgs. \n",
                  (float)(bt2.tms_cstime - bt1.tms_cstime) / _SC_CLK_TCK);
        exit(0);
     }
}
```

Para probar el programa anterior se midió el tiempo de ejecución de los comandos *who* y *date*. Los resultados se presentan a continuación:

```
rogomez@armagnac:219>cc tiempo.c -o tiempo
rogomez@armagnac:220>tiempo date
```

```
Se dan 100 tics por segundo
Thu Mar 11 17:44:46 CST 1999
Tiempos de ejecucion:
        Real: 0.666667 sgs.
        En modo usuario: 0 sgs.
        En modo kernel: 0.666667 sgs.
rogomez@armagnac:221>tiempo who
Se dan 100 tics por segundo
rogomez    console      Mar  8 11:03 (:0)
rogomez    pts/4        Mar  8 11:04
rogomez    pts/5        Mar  8 11:04
rogomez    pts/6        Mar  8 11:04
rogomez    pts/7        Mar  8 11:04
rogomez    pts/8        Mar  8 11:04
rogomez    pts/9        Mar  8 11:04
Tiempos de ejecucion:
        Real: 0.666667 sgs.
        En modo usuario: 0.666667 sgs.
        En modo kernel: 0 sgs.
rogomez@armagnac:222>
```

El programa puede ser usado, no solo para medir el tiempo de ejecución de un comando, sino tambien el de un programa. Esto se muestra a continuación:

```
rogomez@armagnac:225>cat algo.c

#include <stdio.h>
#include <math.h>

main()
{
   int n,i;
   float res;

   for(i=0; i<10000; i++) {
      n=(int)(n*i/(i+1293));
      n=(n*n*n*n*n)/(2);
      res=sqrt(25000);
   }

}
rogomez@armagnac:226>cc algo.c -lm -o algo
rogomez@armagnac:227>tiempo algo
Se dan 100 tics por segundo
Tiempos de ejecucion:
        Real: 0.666667 sgs.
        En modo usuario: 0.333333 sgs.
        En modo kernel: 0.333333 sgs.
rogomez@armagnac:228>
```

En este caso el programa a medir realiza como cálculo una iteración de 10,000 ciclos dentro de la cual realiza operaciones aritméticas (multiplicaciones, divisiones y raíz cuadrada).

Referencias

[1] Deitel H.M. y Deitel P.J. - *Como Programar en C/C++* Ed. Prentice Hall, 1995, 2da. edición

[2] García Márquez Fco. - *Unix Programación Avanzada* Addison-Wesley Iberoamericana, 1994

[3] Stevens Richard W. - *Advanced Programming in the Unix Environnenment* Addison-Wesley Professional Computing Series, 1992

[4] Stevens Richard W. - *Unix Network Programming* Ed. Prentice Hall, 1992

[5] Andrew S. Tanenbaum *Sistemas Operativos Modernos* Ed. Prentice Hall, 1983