

CRONOMETRAJE Y PERFILADO - CRONOMETRAJE*

José Enrique Alvarez Estrada

Translated By:

José Enrique Alvarez Estrada

Based on *Timing and Profiling - Timing*[†] by

Charles Severance

Kevin Dowd

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License [‡]

Vamos a asumir que su programa se ejecuta correctamente. Resulta algo ridículo cronometrar un programa que no se está ejecutando bien, lo cual no quiere decir que no suceda. Dependiendo de lo que esté usted haciendo, puede estar interesado en saber cuánto tiempo gasta globalmente, o interesado sólo en una porción del programa. Le mostraremos cómo cronometrar primero el programa completo, y luego hablaremos acerca de cronometrar bucles o subrutinas individuales.

1 Cronometrando un Programa Completo

En UNIX, puede usted cronometrar la ejecución de un programa poniendo el comando *time* antes que cualquier otro que normalmente teclee en la línea de comandos. Cuando el programa termina, se produce un sumario de cronometraje. Por ejemplo, si su programa se llama *foo*, puede cronometrar su ejecución tecleando `time foo`. Si está usted usando el shell C o el shell Korn, *time* es uno de los comandos internos del shell. Con el shell Bourne, *time* es un comando separado, ejecutado desde */bin*. En cualquier caso, aparece la siguiente información al final de la ejecución:

- Tiempo en modo usuario
- Tiempo en modo sistema
- Tiempo transcurrido

Estas figuras de cronometraje resultan más fáciles de entender con algo de conocimiento previo. Conforme su programa se ejecuta, conmuta de ida y vuelta entre dos modos fundamentalmente diferentes: el *modo de*

*Version 1.2: Oct 19, 2011 12:01 pm -0500

[†]<http://cnx.org/content/m33706/1.3/>

[‡]<http://creativecommons.org/licenses/by/3.0/>

usuario y el *modo de kernel*. El estado de operación normal es el modo de usuario, en el cual se ejecutan las instrucciones que el compilador generó por usted, además de cualquier llamada a una subrutina de biblioteca enlazada con su programa.¹ Pudiera ser suficiente con la ejecución en modo de usuario siempre, excepto que los programas generalmente requieren otros servicios, tales como entrada/salida (I/O), y ello requiere la intervención del sistema operativo -el núcleo o kernel. Una solicitud de servicio del kernel hecha por su programa, o tal vez un evento externo al programa, causa la conmutación del modo de usuario al modo de kernel.

El tiempo empleado en la ejecución de cada uno de los dos modos se contabiliza por separado. La métrica del *tiempo de usuario* describe el tiempo gastado en el modo de usuario. Similarmente, la métrica de *tiempo de sistema* indica el tiempo gastado en modo de kernel. Conforme avanza el tiempo de usuario, cada programa en la máquina se contabiliza por separado. Esto es, no le contabilizarán a usted la actividad realizada por las aplicaciones de alguien más. El registro del tiempo de sistema funciona casi de la misma forma; sin embargo, bajo ciertas circunstancias, puede ser que le contabilicen a usted algunos servicios de sistema realizados a nombre de otras personas, además del suyo propio. Esta contabilidad incorrecta ocurre porque el programa de usted puede estar en ejecución al momento que alguna actividad externa causa una interrupción. No parece justo, pero consuéllese con el hecho de que esto funciona así en ambos sentidos: puede que a los otros usuarios también les contabilicen la actividad de sistema de usted, por la misma razón.

Juntos, los tiempos de usuario y de sistema se conocen como *tiempo de CPU*. Generalmente, el tiempo de usuario es por mucho mayor que el tiempo de sistema. Esto es algo que cabe esperar, porque la mayoría de las aplicaciones sólo piden los servicios del sistema ocasionalmente. De hecho, un tiempo de sistema desproporcionadamente grande probablemente indique algún problema. Por ejemplo, aquellos programas que generan condiciones de excepción repetidamente, tales como fallos de página, referencias a memoria desalineadas, o excepciones de punto flotante, usan una cantidad de tiempo de sistema poco usual. El tiempo gastado en hacer cosas tales como buscar en disco, rebobinar una cinta, o esperar por caracteres provenientes de la terminal no se contabilizan en el tiempo de CPU. Ello se debe a que tales actividades no requieren de la CPU, que está disponible para suspenderlo y ejecutar otros programas.

La tercera pieza de información (correspondiente al tercer conjunto de manecillas del reloj) el *tiempo transcurrido*, es una medida del tiempo actual (tiempo de reloj de pared) que ha pasado desde que el programa inició. Para aquellos programas que gastan la mayoría de su tiempo calculando, el tiempo transcurrido debe ser muy parecido al tiempo de CPU. Las razones por las cuales el tiempo transcurrido puede ser mayor son:

- Está usted compartiendo el tiempo de máquina con otros programas.²
- Su aplicación lleva a cabo mucha I/O
- Su aplicación requiere más ancho de banda de memoria que la que está disponible en la máquina.
- Su programa está llevando a cabo paginación o intercambio.

La gente a menudo registra el tiempo de CPU y lo usa como un estimado del tiempo transcurrido. Usar el tiempo de CPU está bien cuando se trata de máquinas de una sola CPU, suponiendo que ha visto usted ejecutarse el programa cuando la máquina estaba tranquila y observó que ambos números eran muy parecidos. Pero para los multiprocesadores, el tiempo total de CPU puede ser muy diferente del tiempo transcurrido. Cada vez que haya dudas, espere hasta que tenga la máquina sólo para usted, y entonces cronometre su programa, usando el tiempo transcurrido. Es muy importante producir resultados de cronometría que puedan verificarse usando otra corrida, cuando se estén usando los resultados para tomar decisiones de compra importantes.

Si está ejecutando un UNIX derivado de Berkeley, el comando de cronómetro interno del shell C puede reportar otras estadísticas útiles. La forma por defecto de la salida que arroja se muestra en Figure 1 (La función interna *time* de *cs*h). Revise la página del manual de su *cs*h para ver más posibilidades.

Además de los datos de tiempos de CPU y utilizado, el comando *time* de *cs*h produce información acerca del uso de la CPU, fallos de página, intercambios, operaciones de E/S bloqueadas (usualmente actividad de

¹El tiempo de falla de caché también se consume aquí.

²El comando *uptime* le proporciona una indicación aproximada del resto de la actividad en su máquina. Los últimos tres campos le indican el número promedio de procesos listos para ejecutarse durante los últimos 1, 5 y 15 minutos respectivamente

disco) y algunas medidas sobre cuánta memoria ocupaba nuestro programa cuando estaba en ejecución. Las describiremos una a la vez.

1.1 Porcentaje de Uso

El *porcentaje de uso* corresponde a la tasa de tiempo usado respecto al tiempo de CPU. Como mencionamos con anterioridad, existen varias razones por las que el uso de la CPU no sea del 100% o siquiera cercano. A menudo puede usted obtener un indicio, a partir de los otros campos, de cuál es el problema con su programa o si éste estaba compartiendo la máquina cuando lo ejecutaba.

1.2 Uso Promedio de Memoria Real

Las dos medidas de *uso promedio de la memoria* mostradas en Figure 1 (La función interna `time` de `cshtime`) caracterizan los requerimientos de recursos del programa conforme se ejecuta.

La primera medida, el *espacio de memoria compartida*, contabiliza la cantidad promedio de memoria real que ocupa el segmento de texto de su programa - la porción que almacena las instrucciones de máquina. Se le llama "compartida" porque pueden compartirlo varias copias del programa que estén actualmente en ejecución (para ahorrar memoria). Años atrás, era posible que el segmento de texto consumiera una parte significativa del sistema de memoria, pero en estos días, con tamaños de memoria a partir de los 32 MB, tiene usted que compilar un programa fuente realmente enorme y usar cada bit de él para que la cantidad que figure en el uso de memoria compartida sea suficientemente grande como para causar preocupación. El requerimiento de espacio de memoria compartida es usualmente muy bajo, en comparación a la cantidad de memoria disponible en su máquina.

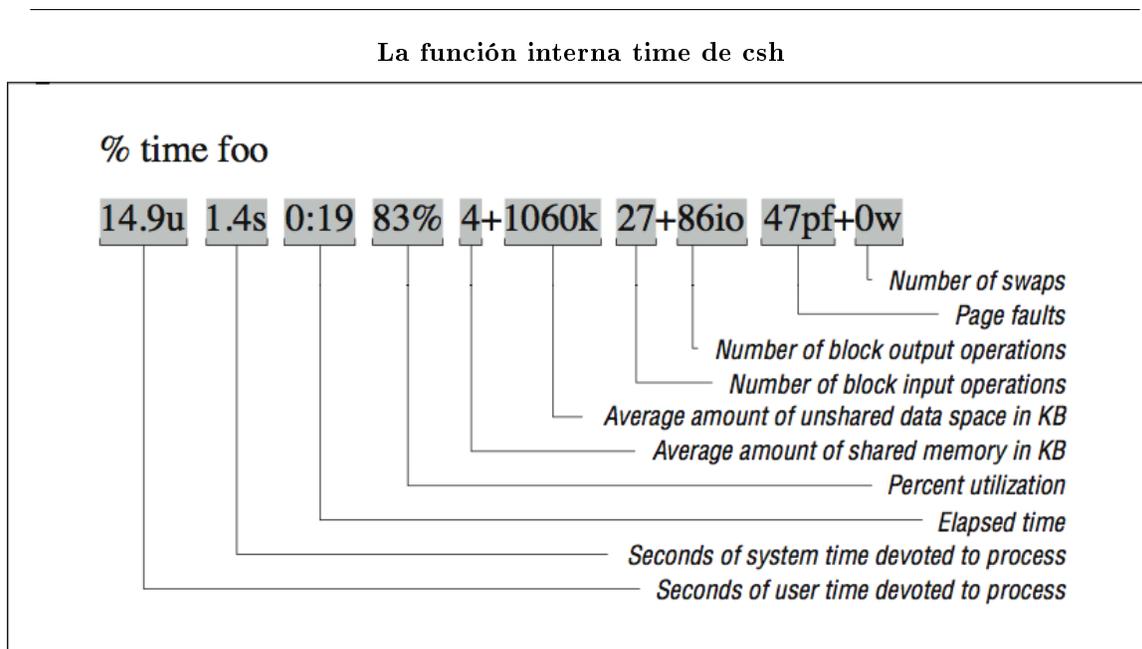


Figure 1

La segunda medida promedio de uso de memoria, el *espacio de memoria no compartida*, describe el almacenamiento *real* dedicado a las estructuras de datos de su programa conforme se ejecuta. Este almacenamiento incluye las variables locales guardadas y las declaradas COMMON para FORTRAN, y las estáticas y externas de C. Resaltamos la palabra "real" aquí y arriba porque estos números nos hablan acerca del uso de memoria física, tomados a lo largo del tiempo. Puede ser que usted haya apartado arreglos con 1 trillón de elementos (espacio virtual), pero si su programa sólo se mueve sobre una esquina de tal espacio, sus requerimientos de memoria a tiempo de ejecución serán muy bajos.

Lo que no le dicen las mediciones de espacio de memoria no compartido, desafortunadamente, es la demanda pico de memoria de su programa. Una aplicación que requiere 100 MB durante 1/10 del tiempo y 1 KB el resto del tiempo parece requerir sólo 10 MB en promedio - lo cuál no es una imagen muy reveladora de los requerimientos de memoria del programa.

1.3 Operaciones de E/S Bloqueadas

Los dos datos para operaciones de *E/S bloqueadas* describen primordialmente el uso de disco, aunque los dispositivos de cintas y algunos otros periféricos pueden también usarse con E/S bloqueada. Las operaciones de E/S de caracteres, como la entrada y salida de la terminal, no aparecen aquí. Un gran número de operaciones de E/S bloqueadas puede explicar un uso de CPU menor del esperado.

1.4 Fallos de Página e Intercambios

Un número inusualmente alto de *fallos de página* o cualquier intercambio probablemente indica un sistema necesitado de memoria, lo cuál también puede explicar un tiempo transcurrido más largo de lo esperado. Puede ser que otros programas estén compitiendo por el mismo espacio. Y no olvide que incluso bajo condiciones óptimas, cada programa sufre de cierto número de fallos de página, como se explicó en here³. En here⁴ se describen algunas técnicas para minimizar el número de fallos de página.

2 Cronometrando una Porción del Programa

Para algunos bancos de pruebas o esfuerzos de afinación, las medidas tomadas desde "afuera" del programa le indican a usted todo lo que necesita saber. Pero si está tratando de aislar las cifras de rendimiento de bucles o porciones de código individuales, puede que quiera incluir rutinas de cronometraje también al interior. La técnica básica es suficientemente sencilla:

1. Registrar el tiempo antes de comenzar a hacer X.
2. Hacer X.
3. Registrar el tiempo al completar X.
4. Restar el tiempo inicial del tiempo final.

Si, por ejemplo, el trabajo primario de X es calcular la posición de unas partículas, divida el tiempo total para obtener un número de posiciones de partículas por segundo. Pero debe ser cuidadoso: si realiza demasiados llamados a las rutinas de cronometraje, el observador se vuelve parte del experimento. Las rutinas de cronometraje también consumen tiempo, y su sola presencia puede incrementar el número de fallos de cache de instrucciones o la paginación. Por otra parte, usted quiere que X tome una cantidad suficiente de tiempo en ejecutarse, como para que las mediciones sean útiles. Es muy importante poner atención al tiempo entre llamadas al cronómetro, porque el reloj usado por las funciones cronométricas tiene una resolución limitada. Un evento que ocurra dentro de una fracción de segundo es difícil de medir con precisión.

³"Memory - Introduction" <<http://cnx.org/content/m32733/latest/>>

⁴"Loop Optimizations - Introduction" <<http://cnx.org/content/m33728/latest/>>

3 Obteniendo Información de Tiempos

En esta sección, discutiremos métodos para obtener varios valores de cronometraje durante la ejecución de su programa.

Para los programas FORTRAN, una biblioteca de funciones de cronometraje disponible en muchas máquinas se llama *etime*, que toma como argumento un arreglo de dos elementos REAL*4, y llena las celdas con el tiempo de CPU de usuario y el tiempo de CPU de sistema, respectivamente. El valor retornado por la función es la suma de los dos. He aquí un ejemplo de cómo se usa *etime* habitualmente:

```
real*4 tarray(2), etime
real*4 start, finish

start = etime(tarray)
finish = etime(tarray)

write (*,*) 'Tiempo de CPU: ', finish - start
```

No todos los vendedores proporcionan una función *etime*; de hecho, algunos no proporcionan rutinas de cronometraje para FORTRAN en absoluto. Pruébelo primero. Si obtiene un mensaje de símbolo indefinido cuando se enlace el programa, puede usar la siguiente rutina en C, que le proporciona la misma funcionalidad que *etime*:

```
#include <sys/times.h>
#define TICKS 100.

float etime (parts)
struct {
    float user;
    float system;
} *parts;
{
    struct tms local;
    times (&local);
    parts->user= (float) local.tms_utime/TICKS;
    parts->system = (float) local.tms_stime/TICKS;
    return (parts->user + parts->system);
}
```

Hay un par de cosas que debe usted ajustar para hacerlo funcionar. Lo primero, para poder enlazar rutinas en C con rutinas en FORTRAN en su computadora, puede ser que deba agregar un guión bajo (_) tras el nombre de la función. Esto cambia la entrada a `float etime_ (parts)`. Además, puede que deba ajustar el parámetro TICKS. Asumimos que el reloj del sistema tenía una resolución de 1/100 de segundo (cierto en las máquinas Hewlett-Packard para las que se escribió esta versión de *etime*). 1/60 es muy común. En una RS-6000 el número debiera ser 1000. Puede encontrar el valor en un archivo llamado `/usr/include/sys/param.h` en su máquina, o bien determinarlo empíricamente.

Abajo se muestra una rutina en C para recuperar la hora real, llamada *gettimeofday*. Está disponible para su uso ya sea en programas C o FORTRAN, y usa paso de parámetros por valor:

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

void hpcwall(double *retval)
{
    static long zsec = 0;
    static long zusec = 0;
    double esec;
    struct timeval tp;
    struct timezone tzp;

    gettimeofday(&tp, &tzp);

    if ( zsec == 0 ) zsec = tp.tv_sec;
    if ( zusec == 0 ) zusec = tp.tv_usec;

    *retval = (tp.tv_sec - zsec) + (tp.tv_usec - zusec ) * 0.000001 ;
}

void hpcwall_(double *retval) { hpcwall(retval); } /* Otra convención */
```

Dado que a menudo necesitará usted tanto el tiempo de CPU como la hora real, y que continuamente estará calculando la diferencia entre sucesivas llamadas a tales rutinas, puede que quiera escribir una rutina que retorne el tiempo de reloj real y el tiempo de CPU cada vez que sea llamada, como sigue:

```

SUBROUTINE HPCTIM(WTIME,CTIME)
  IMPLICIT NONE
*
  REAL WTIME,CTIME
  COMMON/HPCTIMC/CBEGIN,WBEGIN
  REAL*8 CBEGIN,CEND,WBEGIN,WEND
  REAL ETIME,CSCRATCH(2)
*
  CALL HPCWALL(WEND)
  CEND=ETIME(CSCRATCH)
*
  WTIME = WEND - WBEGIN
  CTIME = CEND - CBEGIN
*
  WBEGIN = WEND
  CBEGIN = CEND

```

END

4 Utilizando la Información de Cronometraje

Puede usted obtener mucha información de las facilidades de cronometraje que le proporciona una máquina UNIX. No sólo puede decir cuánto tiempo lleva realizar cierto trabajo, sino también obtener indicios que le digan si la máquina está operando eficientemente, o si hay algún problema que requiera ser solucionado, tal como una memoria inadecuada.

Una vez que el programa está ejecutándose con todas las anomalías antes explicadas, puede usted registrar el tiempo como una línea base. Si está afinándolo, tal línea base será una referencia con la cuál podrá usted afirmar si el proceso de afinación ha mejorado mucho (o poco) las cosas. Si está realizando un banco de pruebas, puede usar dicha línea base para juzgar cuánto incremento global de rendimiento le está dando una máquina nueva. Pero recuerde observar también las otras cifras - paginación, uso de CPU, etc. ya que pueden diferir de máquina en máquina por razones que no están correlacionadas llanamente con el rendimiento de la CPU. Usted quiere asegurarse de obtener la imagen global.