ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Info gprof

Ángel García Fernández

Mayo de 2012

CAPÍTULO 1

Info gprof

Preámbulo

Este apéndice es una traducción libre realizada por mi a partir del fichero Info del Manual de gprof en inglés.

Me he decidido a realizar esta traducción porque no existe ninguna traducción del Info suficientemente completa en español, hasta ahora. Y para la comprensión completa de esta herramienta (gprof) creo que es imprescindible la lectura de ciertas partes de este Manual.

1.1. Perfilando Un Programa: ¿Dónde Gasta Su Tiempo?

Este manual describe el perfilador GNU, gprof, y como puedes usarlo para determinar qué partes de un programa están tomando más tiempo de ejecución. Asumimos que sabes cómo escribir, compilar y ejecutar programas. GNU gprof fue escrito por Jay Fenlason.

Introducción Qué significa perfilado, y por qué es útil (apartado 1.2, página 2).

Compilar Cómo compilar tu programa para ser perfilado (apartado 1.3, página 3).

1.2. Introducción al Perfilado

El perfilado te permite aprender dónde tu programa gasta el tiempo y qué funciones llaman a qué otras funcionas mientras se ejecutaba. Esta información puede mostrar qué partes de tu programa son más lentas de lo que esperabas, y podrían ser candidatas para ser reescritas para hacer que tu programa se ejecute más rápido. También puede decirte qué funciones se están llamando más o menos veces de las que esperabas. Esto puede ayudarte a encontrar errores que de otro modo no te darías cuenta.

Ya que el perfilador usa información recogida durante la ejecución real de tu programa, se puede usar en programas que son demasiado largos o demasiado complejos para analizarlos leyendo el código fuente. Sin embargo, el modo en que tu programa corra afectará a la información que muestra en los datos de perfilado. Si no usas alguna característica de tu programa mientras se está haciendo el perfilan, no se generará información para esa característica.

El perfilado tiene algunos pasos:

- Debes compilar y enlazar tu programa con el perfilado habilitado (apartado 1.3, página 3).

- Debes ejecutar tu programa para generar el fichero de datos de perfilado (apartado 1.4, página 5).
- Debes correr gprof para analizar los datos de perfilado (apartado 1.5, página 6).

Los siguientes tres capítulos explican estos pasos en mayor detalle.

Algunas formas de salida están disponibles a partir del análisis.

El "perfil plano"muestra cuánto tiempo tu programa gasta en cada función, y cuantas veces esa función fue llamada. Si quieres simplemente saber qué funciones queman la mayoría de los ciclos, se exponen concisamente aquí (apartado 1.6.1, página 17).

El "grafo de llamadas"muestra, para cada función, qué funciones la llaman, qué funciones llama, y cuantas veces. Hay también una estimación de cuanto tiempo se gastó en las subrutinas de cada función. Esta puede sugerir lugares donde podrías intentar eliminar llamadas a funciones que usan mucho tiempo (apartado 1.6.2, página 20).

El listado "fuente anotado.es una copia del código fuente del programa, etiquetado con el número de veces que cada línea del programa se ejecutó (apartado 1.6.4, página 29).

Para entender mejor cómo funciona el perfilado, puedes desear leer una descripción de su implementación (apartado 1.9.1, página 35).

1.3. Compilar un Programa para Hacer el Perfilado

El primer paso para generar información de perfilado para tu programa es compilarlo y enlazarlo con el perfilado habilitado.

Para compilar un fichero fuente para perfilado, especifica la opción '-pg' cuando lances el compilador. (Esta se añade a las opciones que normalmente usas.)

Para enlazar el programa para perfilado, si usas un compilador como 'cc' para hacer el enlazado, simplemente especifica '-pg' además de tus opciones usuales. La misma opción, '-pg', altera ambos, la compilación o el enlazado para hacer lo que es necesario para hacer el perfilado. Aquí hay ejemplos:

```
cc -g -c myprog.c utils.c -pg
cc -o myprog myprog.o utils.o -pg
```

La opción '-pg' también funciona con un comando que a la vez compila y enlaza:

cc -o myprog myprog.c utils.c -g -pg

Nota: La opción '-pg' debe ser parte de tus opciones de compilación así como de tus opciones de enlazado. Si no, entonces no se encontrarán los datos del grafo de llamadas y cuando corras gprof tendrás un mensaje de error como este:

```
gprof: gmon.out file is missing call-graph data
```

Si añades la opción '-Q' para suprimir la impresión de los datos del grafo de llamadas, entonces aún podrás ver los periodos de tiempo:

Flat profile:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 44.12      0.07     0.07                             zazLoop
 35.29      0.14     0.06                             main
 20.59      0.17     0.04                             bazMillion

 %         the percentage of the total running time of the
```

Si corres el enlazador 'ld' directamente en lugar de a través de un compilador como 'cc', podrás tener que especificar un un fichero de inicio de perfilado 'gcrt0.o' como el primer fichero de entrada en lugar del habitual 'crt0.o'. Además, querrías probablemente especificar la librería de C de perfilado, 'libc_p.a', escribiendo '-lc_p' en lugar del habitual '-lc'. Esto no es absolutamente necesario, pero haciendo esto te da información del número de llamadas de las funciones de la librería estándar tal como 'read' y 'open'. Por ejemplo:

```
ld -o myprog /lib/gcrt0.o myprog.o utils.o -lc_p
```

Si compilas sólo algunos módulos del programa con '-pg', aún puedes hacer el perfilado del programa, pero conseguirás una información completa sobre los módulos que se compilaron sin '-pg'. La única información que obtienes para las funciones de esos módulos es el tiempo total gastado en ellas; no hay registro de cuantas veces se llamaron, o desde dónde. Esto no afectará el perfil plano (excepto que el campo 'calls' -llamadas- para las funciones se quedará en blanco), pero reducirá enormemente la utilidad del grafo de llamadas.

Si deseas desarrollar un perfilado linea-a-linea, también necesitarás especificar la opción '-g', esto indica al compilador que inserte símbolos de depuración al programa que compara las direcciones del programa con las líneas de código fuente (apartado 1.6.3, página 28).

Además de las opciones '-pg' y '-g', versiones anteriores de GCC requieren que especifiques la opción '-a' cuando compilas con el fin de instrumentalizarlo para realizar el cálculo de los bloques-básicos. Las nuevas versiones no requieren esta opción y no la aceptaran; el cálculo de bloques-básicos estará siempre activado cuando lo está la opción '-pg'.

Cuando el cálculo de bloques-básicos está activado, a medida que el programa corre contará cuantas veces ejecutó cada bifurcación de cada sentencia 'if', cada iteración de cada bucle 'do', etc. Esto habilitará gprof para que construya una lista con el código fuente etiquetado con las veces que cada línea de código fue ejecutada.

No importa que GCC soporte diferentes métodos de perfilado los cuales son habilitados con los parámetros '-fprofile-arcs', '-ftest-coverage' y '-fprofile-values'. Estos parámetros no producen datos que sean útiles para gprof de todos modos, así que no se tratan más en este manual. También hay un parámetro '-finstrument-functions' el cual puede causar que GCC inserte llamadas a rutinas de intrumentación realizadas por el usuario a la entrada y la salida de todas las funciones de su programa. Esto se puede usar para implementar un esquema de perfilado alternativo.

1.4. Ejecutar el programa

Una vez que se ha compilado el programa para perfilarlo, debes correrlo para que genere la información que gprof necesita. Simplemente corre el programa como de costumbre, usando los argumentos, nombre de ficheros, etc que uses normalmente. El programa, sin embargo, correrá algo más lento de lo normal debido al tiempo gastado reuniendo y escribiendo los datos de perfilado.

El modo en que corras el programa–los argumentos y entrada que le des– podría tener un efecto drástico en la información de perfilado mostrada. Los datos de perfilado describirán las partes del programa que se activaron por el uso particular de la entrada que usaste. Por ejemplo, si el primer comando que diste a tu programa es salir, los datos de perfilado sólo mostrarán el tiempo usado en inicializarse y en limpiarse, pero no mucho más.

Tu programa escribirá los datos de perfilado en un fichero llamado 'gmon.out' justamente antes de salir. Si ya hay un fichero llamado

'gmon.out', sus contenidos son sobreescritos. No hay manera actualmente para decirle al programa que escriba los datos de perfilado con otro nombre diferente, pero tu puedes renombrar el fichero más tarde si te preocupa que pueda ser sobreescrito.

A fin de escribir el fichero 'gmon.out' adecuadamente, tu programa debe salir normalmente: por retornar de la función 'main' o por llamar a 'exit'. Llamar a la función de bajo-nivel '_exit' no escribe los datos de perfilado, y tampoco la terminación anómala debido a una señal no manejada.

El fichero 'gmon.out' se escribe en el **directorio de trabajo actual** en el momento que se sale. Esto significa que si tu programa llama a 'chdir', el fichero 'gmon.out' se dejará en el último directorio al que tu programa se cambió mediante 'chdir'. Si no tienes permiso de escritura en este directorio, el fichero no se escribirá, y obtendrás un mensaje de error.

Versiones anteriores de la librería de perfilado GNU puede también escribir un fichero llamado 'bb.out'. Este fichero, si se presenta, contiene una lista con el cálculo de las ejecuciones de los bloques-básicos entendible para los humanos. Desafortunadamente, la aparición de un fichero 'bb.out' entendible para humanos, significa que el cálculo de los bloques-básicos no se escribió en 'gmon.out'. El script Perl 'bbconv.pl', incluido con la distribución fuente de gprof, convertirá un fichero 'bb.out' a un formato entendible para gprof. Invócalo del siguiente modo:

```
bbconv.pl < bb.out > BH-DATA
```

Esto traduce la información de 'bb.out' a una forma que gprof puede entender. Pero tu todavía necesitarás decirle a gprof sobre la existencia de esta información traducida. Para hacer eso, incluye BB-DATA en la línea de comandos de 'gprof', **junto con 'gmon.out'**, de este modo:

```
gprof OPTIONS EXECUTABLE-FILE gmon.out BB-DATA
  [YET-MORE-PROFILE-DATA-FILES...] [> OUTFILE]
```

1.5. Resumen del comando gprof

Después de que tengas el fichero de datos de perfilado 'gmon.out', puedes correr gprof para interpretar la información que hay en él. El programa gprof imprime un perfil plano y un grafo de llamadas en la

salida estándar. Como de costumbre, podrías redireccionar la salida de gprof a un fichero con '>'.

Ejecuta gprof de este modo:

```
gprof OPTIONS [EXECUTABLE-FILE [PROFILE-DATA-FILES...]]
  [> OUTFILE]
```

Aquí entre corchetes se indican los argumentos opcionales.

Si omites el nombre del fichero ejecutable, el fichero que se usa es 'a.out'. Si cualquier fichero no está en el formato adecuado, o si el fichero de datos de perfilado no parece pertenecer al fichero ejecutable, un mensaje de error será imprimido.

Puedes dar más de un fichero de datos de perfilado introduciendo todos sus nombres después del nombre del fichero ejecutable; entonces las estadísticas de todos los ficheros de datos son sumados todos juntos.

El orden de estas opciones no importa.

Opciones de Salida: Controlan el diseño de la salida de gprof (apartado 1.5.1, página 7).

Opciones de Análisis: Controlan como gprof analiza sus datos (apartado 1.5.2, página 12).

Opciones Varias (apartado 1.5.3, página 14).

Opciones Obsoletas Opciones que no se necesitas usar más, pero que se mantienen por compatibilidad (apartado 1.5.4, página 15).

Symspecs Especifican funciones a incluir o excluir (apartado 1.5.5, página 16).

1.5.1. Opciones de Salida

Estas opciones especifican cual de varios formatos de salida 'gprof' debería producir.

Muchas de estas opciones toman un "symspec.ºpcional para especificar las funciones que se incluyen o excluyen. Estas opciones se pueden especificar muchas veces, con diferentes symspecs, para incluir o excluir un conjunto de símbolos (apartado 1.5.5, página 16).

Especificar cualquiera de estas opciones sobreescriben las opciones por defecto ('-p -q'), las cuales imprimen el análisis del perfil plano y del grafo de llamadas para todas las funciones.

‘-A[SYMSPEC]’
‘--annotated-source[=SYMSPEC]’

La opción ‘-A’ causa que gprof imprima el código fuente anotado. Si se especifica SYMSPEC, imprime sólo los símbolos que concuerden (apartado 1.6.4, página 29).

‘-b’
‘--brief’

Si se da la opción ‘-b’, gprof no imprime las notas detalladas que intentan explicar el significado de todos los campos de las tablas. Esto es útil si deseas imprimir la salida, o si estás cansado de ver las notas.

‘-C[SYMSPEC]’
‘--exec-counts[=SYMSPEC]’

La opción ‘-C’ provoca que gprof imprima un recuento de las funciones y el número de veces que cada una fue llamada. Si se especifican SYMSPEC, imprime un recuento sólo para los símbolos que concuerdan.

Si el fichero de datos de perfilado contiene un registro del cálculo de bloques-básicos, especificar la opción ‘-l’, junto con ‘-C’, causará que el cálculo de ejecuciones sea contado y mostrado.

‘-i’
‘--file-info’

La opción ‘-i’ provoca que gprof muestre un resumen de información sobre el(los) fichero(s) de datos de perfilado y después sale. El número registros contabilizados de histogramas, grafos de llamadas, y bloques-básicos.

‘-I DIRS’
‘--directory-path=DIRS’

La opción ‘-I’ especifica una lista de directorios de búsqueda en dónde encontrar los ficheros fuente. La variable de entorno GPROF_PATH también se puede usar para dar esta información. Se usa generalmente para la salida del fuente anotado.

‘-J[SYMSPEC]’
‘--no-annotated-source[=SYMSPEC]’

La opción ‘-J’ provoca que gprof no imprima el código fuente anotado. Si se especifica SYMSPEC, gprof imprimirá el fuente anotado, pero excluirá los símbolos que concuerden.

‘-L’
‘--print-path’

Normalmente, los nombres de los ficheros fuente son imprimidos sin el camino. La opción ‘-L’ provoca que gprof imprima la ruta completa de los nombres de los ficheros fuente, lo que se determina a partir de la información simbólica de depuración en el fichero imagen y es relativo al directorio en el que el compilador fue invocado.

‘-p[SYMSPEC]’
‘--flat-profile[=SYMSPEC]’

La opción ‘-p’ provoca que gprof imprima el perfil plano. Si se especifica SYMSPEC, imprime el perfil plano sólo para los símbolos que concuerdan (apartado 1.6.1, página 17).

‘-P[SYMSPEC]’
‘--no-flat-profile[=SYMSPEC]’

La opción ‘-P’ provoca que gprof no imprima el perfil plano. Si se especifica SYMSPEC, gprof imprime el perfil plano, pero excluye los símbolos que concuerdan.

‘-q[SYMSPEC]’
‘--graph[=SYMSPEC]’

La opción ‘-q’ provoca que gprof imprima el grafo de llamadas. Si se especifica SYMSPEC, imprime el grafo de llamadas sólo para los símbolos que concuerdan y sus hijos (apartado 1.6.2, página 20).

‘-Q[SYMSPEC]’
‘--no-graph[=SYMSPEC]’

La opción ‘-Q’ provoca que gprof no imprima el grafo de llamadas. Si se especifica SYMSPEC, gprof imprime el grafo de llamadas, pero excluye los símbolos que concuerdan.

‘-t’
‘--table-length=NUM’

La opción ‘-t’ provoca que las NUM líneas fuente más activas de cada fichero fuente sean listadas cuando está habilitado el fuente anotado. Por defecto muestra 10.

‘-y’
‘--separate-files’

Esta opción afecta solo a la salida fuente anotada. Normalmente, gprof imprime la salida fuente anotada en la salida-estándar. Si se especifica esta opción, se genera un fichero llamado ‘FILENAME-ann’ para cada fichero llamado ‘path/FILENAME’.Si el sistema de ficheros subyacente truncase ‘FILENAME-ann’ de manera que sobreescribiera al ‘FILENAME’ original, gprof genera en su lugar un fuente anotado en el fichero ‘FILENAME.ann’ (si el nombre del fichero original tuviera extensión, esa extensión sería **reemplazada** por ‘.ann’).

‘-Z[SYMSPEC]’
‘--no-exec-counts[=SYMSPEC]’

La opción ‘-Z’ provoca que gprof no imprima un recuento de las funciones y el número de veces que cada una fue llamada. Si se especifica SYMSPEC, imprime el recuento, pero excluye los símbolos que concuerdan.

‘-r’
‘--function-ordering’

La opción ‘–function-ordering’ provoca que gprof imprima el orden de funciones que sugiere para el programa, basado en los datos de perfilado. Esta opción sugiere un orden en el cual se pueda mejorar la paginación, tlb y el comportamiento de la caché para el programa para sistemas que soporten un orden arbitrario para las funciones en un ejecutable.

Los detalles exactos sobre cómo forzar al enlazador para colocar las funciones en un orden concreto, depende del sistema y está fuera del alcance de este manual.

‘-R MAP_FILE’
‘--file-ordering MAP_FILE’

La opción ‘–file-ordering’ provoca que gprof imprima una orden de de línea para el enlazador .o que sugiere para el programa, basada en los datos de perfilado. Esta opción sugiere una orden que puede mejorar la paginación, tlb y el comportamiento de la caché para el programa para sistemas que no soporten un orden arbitrario para las funciones en un ejecutable.

El uso del argumento ‘-a’ está altamente recomendado con esta opción.

El argumento MAP_FILE es la ruta hacia el fichero que provee el nombre de la función a las asignaciones de ficheros objetivo. El formato del fichero es similar a la salida del programa ‘nm’.

```
c-parse.o:00000000 T yyparse
c-parse.o:00000004 C yyerrflag
c-lang.o:00000000 T maybe_objc_method_name
c-lang.o:00000000 T print_lang_statistics
c-lang.o:00000000 T recognize_objc_keyword
c-decl.o:00000000 T print_lang_identifier
c-decl.o:00000000 T print_lang_type
```

Para crear un MAP_FILE con GNU ‘nm’, escribe un comando parecido a ‘nm –extern-only –defined-only -v –print-file-name program-name’.

‘-T’
‘--traditional’

La opción ‘-T’ provoca que gprof imprima su salida en un estilo BSD "tradicional".

‘-w WIDTH’
‘--width=WIDTH’

Fija la anchura de las líneas de salida a WIDTH. Actualmente sólo se usa cuando imprimes la función índice al final del grafo de llamadas.

‘-x’
‘--all-lines’

Esta opción sólo afecta a la salida fuente anotada. Por defecto sólo las líneas al principio de un bloque-básico se etiquetan. Si esta opción se especifica, todas las líneas en un bloque-básico se etiqueta repitiendo la etiqueta del primer bloque-básico. Este comportamiento es parecido a la opción ‘-a’ de ‘tcov’.

‘--demangle[=STYLE]’
‘--no-demangle’

Estas opciones controlan si deben ser recuperados los nombres originales de C++ cuando se imprime, o no. Por defecto se recuperan. La opción ‘no–demangle’ se puede usar para desactivar la recuperación. Cada compilador tiene un estílo diferente para modificar los nombres. El argumento opcional de estilo de recuperación de nombres, se puede usar para elegir el estilo de recuperación de nombres apropiado para tu compilador.

1.5.2. Opciones de Análisis

‘-a’
‘--no-static’

La opción ‘-a’ provoca que gprof suprima la impresión de las funciones declaradas estáticamente (privadas). (Estas son funciones cuyos nombres no se listan como globales, y los cuales no son visibles fuera del fichero/función/bloque donde fueron definidos.) El tiempo gastado en estas funciones, las llamadas hacia/desde ellas, etc, serán todas atribuidas a la función que fue cargada directamente antes que ella en el fichero ejecutable. Esta opción afecta a ambos, al perfil plano y al grafo de llamadas.

‘-c’
‘--static-call-graph’

La opción ‘-c’ provoca que el grafo de llamadas del programa sea aumentado por un heurístico que examina el espacio del texto del fichero objeto e identifica las llamadas de función en el código de máquina binario. Dado que el registro del grafo de llamadas normal sólo se genera cuando las funciones son llamadas, esta opción identifica los hijos que podrían haber sido y no fueron. Llamadas a funciones que no fueron compiladas con el perfilado habilitado, también se identifican pero sólo si se aparecen en las entradas de la tabla de símbolos. Llamadas a rutinas de una librería dinámica **no** se encuentran por esta opción. Los padres o los hijos identificados por este heurístico se indican en el grafo de llamadas con el contador de llamadas a ‘0’.

‘-D’
‘--ignore-non-functions’

La opción '-D' provoca que gprof ignore aquellos símbolos que no son funciones. Esta opción dará unos datos de perfilado más precisos en aquellos sistemas dónde se soporta (Solaris y HPUX por ejemplo).

`'-k FROM/TO'`

La opción '-k' te permite borrar del grafo de llamadas cualquier arco de los símbolos que concuerdan con symspec FROM de aquellos que concuerdan con symspec TO.

`'-l'`
`'--line'`

La opción '-l' habilita el perfilado línea-a-línea, lo que provoca que los sucesos del histograma se carguen en las líneas de código fuente individuales, en lugar de a funciones. Si el programa se compiló con el recuento de bloques-básicos habilitado, esta opción identificará también cuantas veces cada línea de código se ejecutó. A pesar de que el perfilado línea-a-línea puede ayudar a aislar dónde en una función larga, el programa está gastando su tiempo, también incrementa significativamente el tiempo de ejecución de gprof, y magnifica el error estadístico

`'-m NUM'`
`'--min-count=NUM'`

Esta opción sólo afecta a la salida del conteo de ejecución. Los símbolos que son ejecutados menos de NUM veces se suprimen.

`'-n[SYMSPEC]'`
`'--time[=SYMSPEC]'`

La opción '-n' provoca que gprof, en su análisis del grafo de llamadas, sólo propague los tiempos de los símbolos que concuerden con SYMSPEC.

`'-N[SYMSPEC]'`
`'--no-time[=SYMSPEC]'`

La opción '-n' provoca que gprof, en su análisis del grafo de llamadas, no propague los tiempos de los símbolos que concuerden con SYMSPEC.

‘-z’
‘--display-unused-functions’

Si das la opción ‘-z’, gprof mencionará todas las funciones en el perfil plano, incluidas aquellas que no se llaman nunca, y que no tienen tiempo gastado en ellas. Esto es útil unido a la opción ‘-c’ para descubrir qué rutinas no se llaman nunca.

1.5.3. Opciones varias

‘-d[NUM]’
‘--debug[=NUM]’

La opción ‘-d NUM’ especifica opciones de depurado. Si no se especifica NUM, habilita todo el depurado (apartado 1.9.4, página 43).

‘-h’
‘--help’

La opción ‘-h’ imprime el uso de la línea de comando.

‘-ONAME’
‘--file-format=NAME’

Selecciona el formato del fichero de datos de perfilado. Los formatos que se reconocen son ‘auto’ (por defecto), ‘bsd’, ‘4.4bsd’, ‘magic’, and ‘prof’ (todavía no soportado).

‘-s’
‘--sum’

La opción ‘-s’ provoca que gprof recoger la información en el fichero de datos de perfilado en el que lee, y escribe un nuevo fichero de datos de perfilado llamado ‘gmon.sum’, que contiene la suma de la información de perfil de todos los ficheros de perfil leídos. El fichero ‘gmon.sum’ puede ser uno de los especificados como fichero de entrada; el efecto de esto es combinar los datos de los otros ficheros de entrada dentro de ‘gmon.sum’.

Finalmente, puedes ejecutar gprof sin ‘-s’ para analizar los datos acumulados en el fichero ‘gmon.sum’.

‘-v’
‘--version’

Provoca que gprof muestre el número de versión actual, y después acaba.

1.5.4. Opciones Obsoletas

Estas opciones se han reemplazado por versiones más nuevas que usan symspecs.

`'-e FUNCTION_NAME'`

suprime la impresión de la entrada de perfil de grafo para la rutina FUNCTION_NAME y todos sus descendientes (a menos que tuvieran otros ancestros que no se supriman). Se puede dar más de una opción -e. Sólo un FUNCTION_NAME se puede dar con cada opción -e.

`'-E FUNCTION_NAME'`

suprime la impresión de la entrada de perfil de grafo para la rutina nombre (y sus descendientes) como -e, arriba, y también excluye el tiempo transcurrido en FUNCTION_NAME (y sus descendientes) de los cálculos del total y del porcentaje de tiempo. (Por ejemplo, -E mcount -E mcleanup es lo predeterminado.)

`'-f FUNCTION_NAME'`

imprime la entrada del perfil de grafo de sólo la rutina especi- ficada en FUNCTION_NAME y sus descendientes. Se puede dar más de una opción -f. Sólo se puede dar un FUNCTION_NAME con cada opción -f.

`'-F FUNCTION_NAME'`

imprime la entrada del perfil de grafo de sólo la rutina nombre y sus descendientes (como -f, arriba) y también emplea sólo los tiempos de las rutinas mostradas en los cálculos de total y por- centaje de tiempos. Se puede dar más de una opción -F. Sólo un FUNCTION_NAME se puede dar con cada opción -F. La opción -F tiene preferencia sobre la -E.

Nota que sólo una función se puede especificar con cada opción '-e', '-E', '-f' o '-F'. Para especificar más de una función, usa la opción más de una vez. Por ejemplo, este comando:

```
gprof -e boring -f foo -f bar myprogram > gprof.output
```

lista en el grafo de llamadas todas las funciones que se alcanzaron tanto desde 'foo' como 'bar' y que no se alcanzaron desde 'boring'.

1.5.5. Symspecs

Mucha de las opciones de salida permiten que se incluyan o excluyan funciones usando "symspects" ('symbol specifications'), los que cumplen con las siguiente sintaxis:

```
    nombre_fichero_contiene_un_punto
  | nombre_funcion_no_contiene_un_punto
  | numero_de_linea
  | ( [ cualquier_nombre_fichero ] `:' ( cualquier_nombre_funcion
| numero_de_linea ) )
```

Aquí hay algunos ejemplos de symspecs:

```
`main.c'
```

Selecciona todo lo que esté en el fichero 'main.c'–el punto indica a gprof que interprete esta cadena como un nombre de fichero, en vez de como un nombre de función–. Para seleccionar un fichero cuyo nombre no contenga un punto, se debe especificar terminando el nombre con "dos puntos" (:). Por ejemplo, 'odd:' se interpreta como un fichero de nombre 'odd'.

```
`main'
```

Selecciona todas las funciones de nombre 'main'.

Nota que puede haber muchas instancias con el mismo nombre de función porque algunas de las definiciones pueden ser locales (por ejemplo, static). A menos que el nombre sea único en el programa, debes usar la notación con los "dos puntos" (:) explicada arriba, para especificar una función de un fichero fuente especifico.

Algunas veces, los nombres de funciones contienen puntos. En esos casos, es necesario añadir "dos puntos" (:) al principio del nombre. Por ejemplo, ':.mul' selecciona la función '.mul'.

En algunos formatos de ficheros objeto, los símbolos tienen un subrayado al principio. gprof normalmente no imprimirá esos subrayados. Cuando nombras a un símbolo en un symspec, deberías escribirlo exactamente como lo hace gprof en su salida. Por ejemplo, si el compilador produce un símbolo '_main' de tu función 'main', gprof lo imprime como 'main' en su salida, así que deberías usar 'main' en symspecs.

`'main.c:main'`

Selecciona la función 'main' en el fichero 'main.c'.

`'main.c:134'`

Selecciona la línea 134 del fichero 'main.c'.

1.6. Interpretación de la salida de gprof

gprof puede producir varios estilos de salida diferentes, de los cuales, los más importante se describen a continuación. Los estilos de salida más simples no se describen aquí, pero están documentados en las respectivas opciones que las provocan (apartado 1.5.1, página 7).

Perfil Plano El perfil plano muestra cuanto tiempo de ejecución se gastó directamente en cada función (apartado 1.6.1, página 17).

Grafo de Llamadas El grafo de llamadas muestra qué funciones son llamadas por qué otras, y cuanto tiempo cada función usó cuando fue llamada por éstas.

Line-by-line gprof puede analizar líneas de código fuente individuales

Annotated Source El listado fuente anotado muestra el código fuente etiquetado con un contador de ejecuciones.

1.6.1. El perfil plano

El "perfil plano" muestra la cantidad total de tiempo dedicado a ejecutar su programa de cada función. A menos que la opción '-z' se de, las funciones sin tiempo aparente empleado en ellas, y no hay llamadas aparentes para ellas, no se mencionan. Ten en cuenta que si una función no se compiló para perfilado, y que no se ejecutó el suficiente tiempo para aparecer en el histograma del programa, no será distinguible de una función que nunca fue ejecutada.

Esto es parte de un perfil plano de un pequeño programa:

```
    Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 33.34      0.02     0.02     7208     0.00     0.00  open
 16.67      0.03     0.01      244     0.04     0.12  offtime
 16.67      0.04     0.01        8     1.25     1.25  memccpy
 16.67      0.05     0.01        7     1.43     1.43  write
 16.67      0.06     0.01                             mcount
  0.00      0.06     0.00      236     0.00     0.00  tzset
  0.00      0.06     0.00      192     0.00     0.00  tolower
  0.00      0.06     0.00       47     0.00     0.00  strlen
  0.00      0.06     0.00       45     0.00     0.00  strchr
  0.00      0.06     0.00        1     0.00    50.00  main
  0.00      0.06     0.00        1     0.00     0.00  memcpy
  0.00      0.06     0.00        1     0.00    10.11  print
  0.00      0.06     0.00        1     0.00     0.00  profil
  0.00      0.06     0.00        1     0.00    50.00  report
...
```

Las funciones se ordenan primero, por orden decreciente por tiempo de ejecución empleado en ellas, después decrecientemente, por número de llamadas, y luego alfabéticamente por su nombre. Las funciones 'mcount' y 'profil' son parte del aparato de perfilado y aparecen en todos los perfiles planos, su tiempo da una medida de la cantidad de sobrecarga debido al perfilado.

Justo antes de los encabezados de columna, aparece una declaración que indica a cuanto tiempo equivale una muestra. Este "período de muestreo" estima el margen de error en cada una de las cifras de tiempo. Una cifra de tiempo que no es mucho más grande que esto no es fiable. En este ejemplo, cada muestra cuenta como 0.01 segundos, lo que sugiere una tasa de 100 Hz de muestreo. El tiempo de ejecución total del programa fue de 0,06 segundos, como se indica en el campo de los segundos acumulados ('cumulative seconds'). Ya que cada muestra cuenta como 0,01 segundos, esto significa que sólo seis muestras fueron tomadas durante la ejecución. Dos de las muestras se produjeron mientras el programa estaba en la función 'open', como se indica en el campo de los segundos propios ('self seconds'). Cada una de las otras cuatro muestras se produjeron en 'offtime', 'memccpy', 'write' y 'mcount'. Ya que sólo se han tomado seis muestras, ninguno de estos valores puede ser considerado como particularmente fiable. En otra ejecución, el campo de los segundos propios ('self seconds') para 'mcount' bien podría ser '0,00' o '0,02' (apartado 1.6.6, página 31).

El resto de funciones del listado (aquellas cuyo campo de segundos propios -'self seconds'- es de '0,00') no aparecen en el histograma. Sin embargo, el grafo de llamadas indica que fueron llamadas, por lo

tanto, se enumeran y se ordenan en orden decreciente por el campo de número de llamadas ('calls'). Es evidente que un tiempo fue gastado en ejecutar estas funciones, pero la escasez de muestras del histograma impide cualquier determinación de la cantidad de tiempo que cada una tomó.

Aquí están los significados de los campos de cada línea:

`'% time'`

Este es el porcentaje del tiempo total de ejecución que su programa dedicó a esta función. Estos deben sumar el 100 % entre todos.

`'cumulative seconds'`

Este es el número total de segundos acumulados que el ordenador dedicó a ejecutar estas funciones, más el tiempo dedicado a todas las funciones anteriores a esta en esta tabla.

`'self seconds'`

Este es el número de segundos contabilizados a esta función sola. El listado del perfil plano se ordena por este número.

`'calls'`

Este es el número total de veces que se invocó la función. Si la función no se llamó nunca, o el número de veces que fue llamado no se puede determinar (probablemente porque la función no se compiló con el perfilado habilitado), el campo de las llamadas ('calls') permanecerá en blanco.

`'self ms/call'`

Esta cifra representa el número de milisegundos de media gastados en esta función por cada llamada, si esta función es perfilada. De lo contrario, este campo permanecerá en blanco para esta función.

`'total ms/call'`

total de ms / llamada Est cifra representa el número de milisegundos de media gastados en esta función y sus descendientes por cada llamada, si esta función se perfila. De lo contrario, este campo permanecerá en blanco para esta función. Este es el único campo del perfil plano que utiliza el análisis del grafo de llamadas.

'name'

Este es el nombre de la función. El perfil plano está ordenada alfabéticamente por este campo después de los campos 'self seconds' y 'calls'.

1.6.2. El grafo de llamadas

El 'grafo de llamadas' muestra cuanto tiempo tiempo que se gastó en cada función y en sus hijos. A partir de esta información, usted puede encontrar las funciones que, aunque ellas mismas pueden no haber usado mucho tiempo, llamaron a otras funciones que sí que utilizan cantidades inusuales de tiempo.

Aquí hay un ejemplo de grafo de llamadas de un pequeño programa. Este grafo de llamadas proviene de la misma invocación a gprof que el perfil plano del ejemplo del capitulo anterior.

```
granularity: each sample hit covers 2 byte(s) for 20.00% of 0.05 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.05                 start [1]
                0.00    0.05       1/1           main [2]
                0.00    0.00       1/2           on_exit [28]
                0.00    0.00       1/1           exit [59]
-----------------------------------------------
                0.00    0.05       1/1           start [1]
[2]    100.0    0.00    0.05       1         main [2]
                0.00    0.05       1/1           report [3]
-----------------------------------------------
                0.00    0.05       1/1           main [2]
[3]    100.0    0.00    0.05       1         report [3]
                0.00    0.03       8/8           timelocal [6]
                0.00    0.01       1/1           print [9]
                0.00    0.01       9/9           fgets [12]
                0.00    0.00      12/34          strncmp <cycle 1> [40]
                0.00    0.00       8/8           lookup [20]
                0.00    0.00       1/1           fopen [21]
                0.00    0.00       8/8           chewtime [24]
                0.00    0.00       8/16          skipspace [44]
-----------------------------------------------
[4]     59.8    0.01        0.02       8+472     <cycle 2 as a whole>        [4]
                0.01        0.02     244+260         offtime <cycle 2> [7]
                0.00        0.00     236+1           tzset <cycle 2> [26]
-----------------------------------------------
```

Las líneas de guiones dividen esta tabla en 'entradas', una para cada función. Cada entrada tiene una o más líneas.

En cada entrada, la línea principal es la que comienza con un número de índice entre corchetes. Al final de esta línea aparece el nombre de la función a la que esta entrada se refiere. Las líneas anteriores de la entrada describen a los llamadores de esta función y las líneas siguientes describen sus subrutinas (también llamados 'hijos' cuando se habla del grafo de llamadas).

Las entradas se ordenan por el tiempo gastado en la función y sus subrutinas.

La función interna de perfilado 'mcount' (apartado 1.6.1, página 17) no se menciona nunca en el grafo de llamadas.

Línea principal Detalles del contenido de la línea principal.

Llamadores Detalles del contenido de las líneas de llamadores.

Subrutinas Detalles del contenido de las líneas para subrutinas.

Ciclos Cuando hay ciclos recursivos, tales como 'a' llama a 'b' y 'b' llama a 'a'.

LÍNEA PRINCIPAL

La 'línea principal' de la entrada del grafo de llamadas es la línea que describe la función sobre la que trata la entrada y da las estadísticas generales para esta función.

Como referencia, repetimos la línea principal de la entrada para la función 'report' del ejemplo principal, junto con la línea de cabecera que muestra los nombres de los campos:

```
index  % time    self  children called     name
...
[3]    100.0    0.00    0.05       1         report [3]
```

Aquí está lo que significan los campos en la línea principal:

'index'

Las entradas están numeradas con enteros consecutivos. Por lo tanto, cada función tiene un número de índice, que aparece al principio de su línea principal.

Cada referencia cruzada a una función, como un llamador o subrutina de otra, da su número de índice, así como su propio nombre. El número de índice te guía si deseas buscar la entrada de esa función.

`'% time'`

Este es el porcentaje del tiempo total que se gastó en esta función, incluido el tiempo dedicado en las subrutinas llamadas desde esta función.

El tiempo gastado en esta función se cuenta de nuevo para los llamadores de esta función. Por lo tanto, la suma de estos porcentajes no tiene sentido.

`'self'`

Esta es la cantidad total de tiempo dedicado a esta función. Esta debe ser idéntico al número impreso en el campo 'seconds' (segundos) de esta función en el perfil plano.

`'children'`

Esta es la cantidad total de tiempo dedicado a las llamadas a subrutinas realizadas por esta función. Esta debe ser igual a la suma de todos los campos 'self' y 'children' de todos los hijos que aparecen justo debajo de esta función.

`'called'`

Este es el número de veces que se llama a la función.

Si la función se llama de forma recursiva, hay dos números, separados por el signo '+'. El primer número cuenta las llamadas no recursivas, y el segundo cuenta las llamadas recursivas.

En el ejemplo anterior, la función 'report' fue llamada una vez desde 'main'.

`'name'`

Este es el nombre de la función actual. El número de índice se repite después de él.

Si la función es parte de un ciclo recursivo, el número del ciclo se imprime entre el nombre de la función y el número de índice (apartado 1.6.2, página 25). Por ejemplo, si la función 'gnurr' es parte del ciclo número uno, y tiene el número de índice doce, su línea principal sería terminaría así:

```
gnurr <cycle 1> [12]
```

Líneas para las funciones llamadoras

La entrada de una función tiene una línea para cada función por la que fue llamada. Los campos de estas líneas corresponden a los campos de la línea principal, pero sus significados son diferentes debido al diferente contexto.

Como referencia, repetimos dos líneas de la entrada para la función 'report', la línea principal y una línea de llamador precediéndola, junto con la línea de cabecera que muestra los nombres de los campos:

```
  index  % time    self  children called     name
  ...
                 0.00    0.05       1/1           main [2]
  [3]    100.0    0.00    0.05       1         report [3]
```

Aquí están los significados de los campos de una línea de llamadores de la función 'report', llamada 'main':

`'self'`

Una estimación de la cantidad de tiempo dedicada la función 'report' cuando fue llamada por la función 'main'.

`'children'`

Una estimación de la cantidad de tiempo dedicado en las subrutinas de 'report' cuando 'report' fue llamada por 'main'.

La suma de los campos 'self' y 'children' son una estimación de la cantidad de tiempo dedicado en las llamadas a 'report' desde 'main'.

`'called'`

Dos números: el número de veces que 'report' fue llamada desde 'main', seguido por el número total de llamadas no recursivas hechas a 'report' desde todos sus llamadores.

'name and index number'

El nombre de la función que llama a 'report' a la que se aplica esta línea, seguido por su número de índice.

No todas las funciones tienen entradas en el grafo de llamadas; algunas opciones de gprof solicitan la omisión de ciertas funciones. Cuando un función llamadora no tiene su entrada propia, tiene líneas como llamadora en las entradas de las funciones a las llama.

Si la función llamadora forma parte de un ciclo de recursión, el número del ciclo se imprime entre el nombre y el número de índice.

Si la identidad de función llamadora de una función no se puede determinar, una línea llamadora ficticia se imprime y recibe el nombre de <spontaneous>(espontánea) y todos los demás campos en blanco. Esto puede ocurrir por los manejadores de eventos o señales.

LÍNEAS PARA SUBRUTINAS (FUNCIONES LLAMADAS)

La entrada de una función tiene una línea para cada uno de sus subrutinas - en otras palabras, una línea para cada función que se llama. Los campos de estas líneas corresponden a los campos de la línea principal, pero sus significados son diferentes debido al diferente contexto.

Como referencia, repetimos dos líneas de la entrada de la función 'main', la línea principal y una línea de una subrutina, junto con la línea de cabecera que muestra los nombres de los campos:

```
index  % time    self  children called     name
...
[2]    100.0    0.00    0.05       1         main [2]
                0.00    0.05       1/1           report [3]
```

Aquí están los significados de los campos de la línea de subrutina de la llamada de 'main' a 'report':

'self'

Una estimación de la cantidad de tiempo dedicado directamente en 'report' cuando 'report' fue llamada desde 'main'.

`'children'`

Una estimación de la cantidad de tiempo dedicado en las subrutinas de 'report' cuando 'report' fue llamada desde 'main'.

La suma de los campos 'self' y 'children' es una estimación del tiempo total dedicado en las llamadas a 'report' desde 'main'.

`'called'`

Dos números, el número de llamadas hechas a 'report' desde 'main' seguido por el número total de llamadas no recursivas hechas a 'report'. Esta relación se utiliza para determinar la cantidad de tiempo propio ('self') de 'report' y de sus hijos ('children') se le atribuye a la función 'main' (apartado 1.6.7, página 32).

`'name'`

El nombre de la subrutina de 'main' a la que se aplica esta línea, seguido por el número de índice de la subrutina.

Si la función llamadora forma parte de un ciclo de recursión, el número del ciclo se imprime entre el nombre y el número de índice.

CÓMO SE DESCRIBEN LAS FUNCIONES MUTUAMENTE RECURSIVAS

El grafo se puede complicar por la presencia de 'ciclos de recursión' en el grafo de llamadas. Existe un ciclo si una función llama a otra función que (directa o indirectamente) llama a (o parece llamar a) la función original. Por ejemplo: si 'a', llama a 'b', y 'b' llama a 'a', entonces 'a' y 'b' forman un ciclo.

Siempre hay caminos de llamadas en ambos sentidos entre un par de funciones, que pertenecen a un mismo ciclo. Si 'a' y 'b' se llaman entre sí y 'b' y 'c' se llaman entre sí, los tres hacen un ciclo. Tenga en cuenta que incluso si 'b' sólo llama a 'a' si no fue llamada desde 'a', gprof no puede determinar esto, así que 'a' y 'b' se seguirán considerando un ciclo.

Los ciclos se numeran con enteros consecutivos. Cuando una función pertenece a un ciclo, cada vez que el nombre de la función aparece en el grafo de llamadas, le sigue '<cycle N>' (ciclo número N).

El razón por la que los ciclos son importantes es que ellos hacen que el valores de tiempo en el grafo de llamadas sea una paradoja. El

'tiempo dedicado en los hijos' de 'a' debería incluir el tiempo dedicado en su subrutina 'b' y en las subrutinas de 'b' -¡pero una de las subrutinas de 'b' es 'a'! ¿Qué cantidad de tiempo se debe incluir en los hijos de 'a', cuando 'a'es indirectamente recursiva?

La forma en que gprof resuelve esta paradoja es creando una sola entrada para el ciclo en su conjunto. La línea principal de esta entrada describe el tiempo total dedicado directamente en las funciones del ciclo. Las 'subrutinas' del ciclo son las funciones individuales del ciclo, y todas las demás funciones que se llaman directamente por ellas. Las 'funciones llamadoras' del ciclo son las funciones, fuera del ciclo, que llamaron a funciones del ciclo.

He aquí un ejemplo de una parte de un grafo de llamadas que muestra un ciclo que contiene las funciones de 'a' y 'b'. El ciclo ha sido proporcionada por una llamada a 'a' desde 'main'; ambas 'a' y 'b' llamaron a 'c'

```
index  % time    self  children called     name
-----------------------------------------
                1.77        0    1/1        main [2]
[3]     91.71   1.77        0    1+5    <cycle 1 as a whole> [3]
                1.02        0    3          b <cycle 1> [4]
                0.75        0    2          a <cycle 1> [5]
-----------------------------------------
                                 3          a <cycle 1> [5]
[4]     52.85   1.02        0    0      b <cycle 1> [4]
                                 2          a <cycle 1> [5]
                   0        0    3/6        c [6]
-----------------------------------------
                1.77        0    1/1        main [2]
                                 2          b <cycle 1> [4]
[5]     38.86   0.75        0    1      a <cycle 1> [5]
                                 3          b <cycle 1> [4]
                   0        0    3/6        c [6]
-----------------------------------------
```

(El grafo de llamadas completo para este programa contiene además una entrada para 'main', que llama a 'a', y una entrada para 'c', con sus llamadores 'a' y 'b'.)

```
index  % time    self  children called     name
                                            <spontaneous>
[1]    100.00      0     1.93    0      start [1]
                0.16     1.77    1/1        main [2]
-----------------------------------------
                0.16     1.77    1/1        start [1]
[2]    100.00   0.16     1.77    1      main [2]
                1.77        0    1/1        a <cycle 1> [5]
-----------------------------------------
                1.77        0    1/1        main [2]
[3]     91.71   1.77        0    1+5    <cycle 1 as a whole> [3]
                1.02        0    3          b <cycle 1> [4]
```

```
                0.75        0    2          a <cycle 1> [5]
                   0        0    6/6        c [6]
-----------------------------------------
                                 3          a <cycle 1> [5]
[4]     52.85   1.02        0    0      b <cycle 1> [4]
                                 2          a <cycle 1> [5]
                   0        0    3/6        c [6]
-----------------------------------------
                1.77        0    1/1        main [2]
                                 2          b <cycle 1> [4]
[5]     38.86   0.75        0    1      a <cycle 1> [5]
                                 3          b <cycle 1> [4]
                   0        0    3/6        c [6]
-----------------------------------------
                   0        0    3/6        b <cycle 1> [4]
                   0        0    3/6        a <cycle 1> [5]
[6]      0.00       0        0    6      c [6]
-----------------------------------------
```

El campo 'self' de la línea principal del ciclo es el tiempo total gastado en todas las funciones del ciclo. Es igual a la suma de los campos 'self' de las distintas funciones en el ciclo, que se encuentran en la entrada, en las líneas de subrutina para estas funciones.

Los campos 'children' de la línea principal del ciclo y las líneas de subrutina sólo recuentan las subrutinas de fuera del ciclo. A pesar de que 'a' llama a 'b', el tiempo empleado en esas llamadas a 'b' no se puede contar en el tiempo de los hijos de 'a'. Por lo tanto, no nos encontramos con el problema de qué hacer cuando el tiempo en esas llamadas a 'b' incluyen llamadas recursivas indirectas de vuelta a 'a'.

El campo 'children' de una línea de función llamadora en la entrada del ciclo, estima la cantidad de tiempo dedicado en **todo el ciclo**, y sus otras subrutinas, en los tiempos en que la función llamadora llamó a una función en del ciclo.

El campo 'calls' de la línea principal para el ciclo tiene dos números: el primero, el número de veces que las funciones del ciclo fueron llamadas por las funciones de fuera del ciclo; en segundo lugar, el número de veces que fueron llamados por las funciones del ciclo (incluyendo las veces en que una función del ciclo se llama a sí misma). Esta es una generalización de la división habitual de llamadas recursivas y no recursivas.

El campo 'calls' de la línea de una subrutina para un miembro del ciclo en la entrada del ciclo muestra cuantas veces esa función fue llamada por las funciones del ciclo. El total de todos estos es el segundo número en el campo 'calls' de la línea principal.

En la entrada individual para una función de un ciclo, las demás funciones que en el mismo ciclo pueden aparecer como subrutinas y como funciones llamadoras. Estas líneas muestran cuántas veces

cada función del ciclo llamó o fue llamada desde cada función del ciclo. Los campos 'self' y 'children' en estas líneas están en blanco debido a la dificultad de definir el significado para éstos cuando hay recursividad.

1.6.3. Perfilado línea a línea

La opción '-l' de gprof provoca que el programa realice el perfilado 'línea-a-línea'. En este modo, las muestras del histograma no están asignadas a funciones, sino a las líneas individuales del código fuente. El programa por lo general debe ser compilado con la opción '-g', además de '-pg', con el fin de generar símbolos de depuración para el seguimiento de las líneas de código fuente.

El perfil plano es la tabla de salida más útil en el modo de 'línea-a-línea'. El grafo de llamadas no es tan útil como normalmente, ya que la versión actual de gprof no se propaga a los arcos del grafo de llamadas de las líneas del código fuente para la función que la encierra. El grafo de llamada, sin embargo, muestra cada línea de código que llama a cada función, junto con un recuento.

Esta es una sección de la salida de gprof, sin perfiles línea-a-línea. Tenga en cuenta que 'ct_init' representaron cuatro aciertos del histograma, y 13.327 llamadas a 'init_block'.

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 30.77      0.13     0.04     6335     6.31     6.31  ct\_init


                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 7.69\% of 0.13 seconds

index \% time    self  children    called     name

                0.00    0.00       1/13496       name_too_long
                0.00    0.00      40/13496       deflate
                0.00    0.00     128/13496       deflate_fast
                0.00    0.00   13327/13496       ct_init
[7]      0.0    0.00    0.00   13496         init_block
```

Ahora echemos un vistazo a algunas de las salidas de gprof de la mima ejecución de un programa, esta vez con los perfiles línea-a-línea habilitados. Tenga en cuenta que los cuatro aciertos de histograma de 'ct_init' se dividen en cuatro líneas de código fuente - un golpe

ocurrió en cada una de las líneas 349, 351, 382 y 385. En el grafo de llamadas, tenga en cuenta cómo las 13.327 llamadas de 'ct_init' a 'init_block' se dividen en una llamada desde la línea 396, 3.071 llamadas desde la línea 384, 3.730 llamadas desde la línea 385, y 6.525 llamadas desde 387.

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self
 time   seconds   seconds    calls  name
  7.69      0.10     0.01           ct\_init (trees.c:349)
  7.69      0.11     0.01           ct\_init (trees.c:351)
  7.69      0.12     0.01           ct\_init (trees.c:382)
  7.69      0.13     0.01           ct\_init (trees.c:385)


                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 7.69\% of 0.13 seconds

  % time    self  children    called     name

            0.00    0.00       1/13496       name_too_long (gzip.c:1440)
            0.00    0.00       1/13496       deflate (deflate.c:763)
            0.00    0.00       1/13496       ct_init (trees.c:396)
            0.00    0.00       2/13496       deflate (deflate.c:727)
            0.00    0.00       4/13496       deflate (deflate.c:686)
            0.00    0.00       5/13496       deflate (deflate.c:675)
            0.00    0.00      12/13496       deflate (deflate.c:679)
            0.00    0.00      16/13496       deflate (deflate.c:730)
            0.00    0.00     128/13496       deflate_fast (deflate.c:654)
            0.00    0.00    3071/13496       ct_init (trees.c:384)
            0.00    0.00    3730/13496       ct_init (trees.c:385)
            0.00    0.00    6525/13496       ct_init (trees.c:387)
[6]  0.0    0.00    0.00   13496         init_block (trees.c:408)
```

1.6.4. Listado desglosado de los ficheros fuentes

La opción '-A' de gprof activa un listado de código fuente con anotaciones, que muestra el código fuente del programa, con cada función etiquetada con el número de veces que fue llamada. También puede ser necesario especificar la opción '-I', si gprof no puede encontrar los ficheros de código fuente.

Compilar con 'gcc ... -g-pg-a' aumenta su programa con el recuento de bloques-básicos en el código, además de la función del recuento de funciones. Esto permite gprof determinar cuántas veces cada línea de código se ejecutó. Por ejemplo, considere la siguiente función, tomada de 'gzip', con los números de línea agregados:

```
    1 ulg updcrc(s, n)
    2     uch *s;
```

```
 3     unsigned n;
 4 {
 5     register ulg c;
 6
 7     static ulg crc = (ulg)0xffffffffL;
 8
 9     if (s == NULL) {
10         c = 0xffffffffL;
11     } else {
12         c = crc;
13         if (n) do {
14             c = crc_32_tab[...];
15         } while (--n);
16     }
17     crc = c;
18     return c ^ 0xffffffffL;
```

'updcrc' tiene al menos cinco bloques-básicos. Uno de ellos es la propia función. La declaración 'if' en la línea 9 genera dos bloques-básicos más, uno para cada rama del 'if'. Un cuarto bloque-básico se obtiene del 'if' de la línea 13, y el contenido del bucle 'do' forma el quinto bloque-básico. El compilador también puede generar bloques-básicos adicionales para manejar varios casos especiales.

Un programa ampliado con recuento de bloques-básicos se puede analizar con 'gprof -l -A'. También sugiero el uso de la opción '-x', que asegura que cada línea de código se etiqueta al menos una vez. Aquí está el listado del fuente con anotaciones para una ejecución de 'gzip' de muestra:

```
                    ulg updcrc(s, n)
                        uch *s;
                        unsigned n;
                2 ->{
                        register ulg c;

                        static ulg crc = (ulg)0xffffffffL;

                2 ->    if (s == NULL) {
                1 ->        c = 0xffffffffL;
                1 ->    } else {
                1 ->        c = crc;
                1 ->        if (n) do {
            26312 ->            c = crc_32_tab[...];
    26312,1,26311 ->        } while (--n);
                        }
                2 ->    crc = c;
                2 ->    return c ^ 0xffffffffL;
```

En este ejemplo, la función se llama dos veces, pasando una vez por cada rama de la instrucción 'if'. El cuerpo del bucle 'do' se ejecutó un total de 26.312 veces. Nota cómo la declaración 'while' está etiquetada. Se inició la ejecución 26.312 veces, una por cada iteración

del bucle. Una de esas veces (la última vez) salió del bucle, mientras que volvió al principio del bucle 26.311 veces.

1.6.5. Fiabilidad de gprof

Error de muestreo Márgenes de error estadísticos (apartado 1.6.6, página 31).

Supuestos Estimar el tiempo de los hijos (apartado 1.6.7, página 32).

1.6.6. Error estadístico de muestreo

Las cifras que da gprof se basan en un proceso de muestreo y por tanto están sujetas a errores estadísticos. Si una función se ejecuta durante un corto intervalo de tiempo, de modo que en promedio, esperamos que el proceso de muestreo "fotografíe" la función una sola vez, es bastante probable que al final no la "encontremos" o nos la encontremos un par de veces.

En contraste, el número de llamadas y de bloques básicos se calculan mediante contadores, no por muestreo. Son completamente acertados y no varían entre distintas ejecuciones si el programa es determinista.

El "periodo de muestreo" (sampling period) que se imprime al comienzo del perfil plano (Flat Profile) informa acerca de cuan a menudo se toman las muestras. Como regla general, una cifra de tiempo de ejecución es más precisa si es considerablemente mayor que el periodo de muestreo.

El error real puede ser calculado. Para una muestra de tamaño N, el error **esperado** es la raíz cuadrada de N. Por ejemplo, si el periodo de muestreo es 0'01 segundos y la función 'foo' toma 1 segundo, N es 100 muestras (1 segundo/0'01 segundos), la raíz cuadrada de N es 10 muestras, así que el error esperado en 'foo' es de 0'1 segundos (10*0'01 segundos), o el diez por ciento del valor observado. De nuevo, si el periodo de muestreo es 0'01 segundos y el tiempo de ejecución de 'bar' es de 100 segundos, N es 10.000 muestras, la raíz de N es 100 muestras, así que el error esperado en 'bar' es de 1 segundo, o un uno por ciento del valor observado. Es probable que varíe todo esto **de media** del perfil de una ejecución al siguiente (**Algunas veces** variará más).

Esto no significa que un tiempo de ejecución pequeño no sea significativo. Si el tiempo de ejecución del programa es grande, ello indica que la función sólo toma una fracción pequeña del tiempo total. La

consecuencia natural es que, no vale la pena emplear mucho tiempo en optimizar dicha función.

Una forma de incrementar la precisión es darle al programa una entrada (similar) pero que tome mas tiempo. Otra es combinar varias ejecuciones utilizando la opción '-s' de gprof. Este es el método:

1. Ejecuta tu programa una vez.
2. Escribe la orden:

   ```
   mv gmon.out gmon.sum
   ```

3. Ejecuta tu programa de nuevo, igual que antes.
4. Fusiona los datos de 'gmon.out' con los de 'gmon.sum' con este comando:

   ```
   gprof -s EXECUTABLE-FILE gmon.out gmon.sum
   ```

5. Repite estos dos últimos pasos tanto como quieras.
6. Analiza los datos acumulados usando este comando:

   ```
   gprof EXECUTABLE-FILE gmon.sum > OUTPUT-FILE
   ```

1.6.7. Estimar los tiempos de los 'hijos'

Algunas de las cifras del grafo de llamadas, son estimadas –por ejemplo, los valores de tiempo de los 'hijos' y todas las cifras de tiempo en las líneas de llamadores y subrutinas.

No hay información directa sobre estas mediciones en los propios datos de perfilado. En su lugar, gprof los estima haciendo una suposición sobre tu programa podría o no ser verdad.

El supuesto hecho es que el tiempo gastado en cada llamada a cualquier función 'foo' no está relacionada con quien llama a 'foo'. Si 'foo' usa 5 segundos en total, y 2/5 de las llamadas a 'foo' vienen de 'a', entonces se supone que 'foo' contribuye 2 segundos al tiempo en los 'hijos' de 'a'.

Esta suposición es normalmente suficientemente cierta, pero para algunos programas está muy lejos de la verdad. Supón que 'foo' termina muy rápido cuando su argumento es 0; supón que 'a' siempre pasa 0 como argumento, mientras que otros llamadores de 'foo' pasan otros argumentos. En este programa, todo el tiempo gastado

en 'foo' se hace en las llamadas desde otras funcionas que no son 'a'. Pero gprof no tiene forma de saberlo; cargará ciega e incorrectamente 2 segundos de tiempo de 'foo' a los hijos de 'a'.

Esperamos algún día poner información más completa dentro de 'gmon.out', para que esta suposición no sea necesaria nunca más, si podemos imaginar cómo. Por lo pronto, las cifras estimadas son normalmente más útiles que erróneas.

1.7. Respuesta a preguntas frecuentes

¿Cómo puedo obtener información más exacta sobre los puntos calientes de mi programa

Mirar a los contadores por línea sólo te dice parte de la historia. Porque gprof sólo puede reportar los tiempo de las llamadas y los contadores por función, el mejor modo para obtener un a información más nítida sobre dónde el programa está gastando su tiempo es refactorizar las funciones más grandes en secuencias de llamadas a funciones más pequeñas. Ten cuidado con esto porque también puede introducir puntos calientes artificiales puesto que compilar con '-pg' añade un importante incremento a las llamadas de funciones. Una solución alternativa es usar un perfilador no intrusivo como oprofile.

¿Cómo puedo saber qué líneas de mi programa se ejecutaron a la mayoría de las veces?

Compila tu programa con el conteo de bloques-básicos habilitados, y entonces, ejecuta la siguiente tubería:

```
gprof -l -C OBJFILE | sort -k 3 -n -r
```

Este listado te mostrará qué líneas de tu código se ejecutaron más a menudo, pero no necesariamente, aquellas que consumieron la mayor parte del tiempo.

¿Cómo puedo saber qué líneas de mi programa llamaron a una función en particular?

Use 'gprof -l' y la búsqueda de la función en el gráfico de llamadas. Los llamadores se desglosan por función y número de línea.

¿Cómo analizo un programa que tiene una duración de menos de un segundo??

Intente usar un script de shell como este:

```
for i in `seq 1 100`; do
  fastprog
  mv gmon.out gmon.out.$i
done
```

```
gprof -s fastprog gmon.out.*

gprof fastprog gmon.sum
```

Si su programa es completamente determinista, todos los contadores serán simplemente múltiplos de 100 (es decir, una función llamada una vez en cada ejecución aparecerá como llamada 100 veces).

1.8. Incompatibilidades con gprof de Unix

GNU gprof y Berkeley Unix gprof utilizan los mismos datos del archivo gmon.out, y proporcionan básicamente la misma información. Pero hay algunas diferencias.

- GNU gprof utiliza un formato nuevo y generalizado con soporte para recuento de ejecución de bloques-básicos y los histogramas que no son en tiempo real. Una cookie mágica y un número de versión permite a gprof identificar fácilmente los archivos con nuevo estilo. Los archivos al estilo viejo BSD archivos todavía se pueden leer (apartado 1.9.2, página 37).
- Para una función recursiva, Unix gprof muestra la función como padre y como un hijo, con el campo 'calls' que muestra el número de llamadas recursivas. GNU gprof omite estas líneas y coloca el número de llamadas recursivas en la línea principal.
- Cuando una función se suprime del grafo de llamadas con '-e', GNU gprof todavía la enumera como una subrutina de las funciones que la llaman.
- GNU gprof acepta la opción '-k' con su argumento en la forma 'from/to', en lugar de 'from to'.
- En el listado del fuente anotado, si hay varios elementos bloques-básicos en la misma línea, GNU gprof imprime todos sus recuentos, separados por comas.
- Los comentarios, los anchos de los campos, y los formatos de salida son diferentes. GNU gprof imprime comentarios después de las tablas, así puedes ver las tablas sin tener que saltarte los comentarios.

1.9. Detalles del perfilado

Implementación Cómo un programa recopila la información de perfilado (apartado 1.9.1, página 35).

Formato del fichero Formato de los ficheros 'gmon.out' (apartado 1.9.2, página 37)

Interiores Funcionamiento interno de gprof (apartado 1.9.3, página 39)

Depurado Usar la opción '-d' de gprof (apartado 1.9.4, página 43)

1.9.1. Implementación

El perfilado funciona cambiando cómo se compila cada función en tu programa para que cuando es llamada, irá acumulando información acerca de desde donde fue llamada. A partir de esto, el perfilador puede determinar qué función la llamó, y puede contar cuantas veces fue llamada. Este cambio se realiza por el compilador cuando el programa se compila con la opción '-pg', que es la que provoca que cada función llame a 'mcount' (o '_mcount', o '__mcount', dependiendo del sistema operativo y el compilador) como una de sus primeras operaciones.

La rutina 'mcount', incluida en la biblioteca de perfilado, es responsable de registrar en memoria la tabla del grafo de llamadas tanto para la rutina de su padre (el hijo) y el padre de su padre. Esto se suele hacer mediante el examen de la pila del programa para encontrar tanto la dirección del hijo, como la dirección de retorno en el padre original. Dado que esta es una operación muy dependiente de la máquina, 'mcount' sí suele ser una pequeña rutina auxiliar en lenguaje ensamblador que extrae la información requerida, y luego llama a '__mcount_internal' (una función normal de C) con dos argumentos - 'frompc' y 'selfpc'. '__mcount_internal' es responsable de mantener el grafo de llamadas en la memoria, los registros 'frompc', 'selfpc', y el número de veces que cada uno de estos arcos se recorrieron.

GCC versión 2 proporciona una función mágica ('__builtin_return_address'), que permite una función 'mcount' genérica para extraer la información necesaria de la pila del programa. Sin embargo, en algunas arquitecturas, sobre todo en SPARC, utilizar esta orden interna puede ser muy costoso computacionalmente, y una versión en lenguaje ensamblador de 'mcount' se utiliza por razones de rendimiento.

La información del número de llamadas para rutinas de la librería se obtienen mediante el uso de una versión especial de la biblioteca

de C. Los programas son los mismos que en la librería habitual de C, pero compilados con '-pg'. Si enlazas tu programa con 'gcc ... -pg', usará automáticamente la versión de la biblioteca para perfilados.

Perfilar también implica observar tu programa mientras se ejecuta, y mantener un histograma de dónde está el contador del programa de vez en cuando. Normalmente, el contador de programa se ve alrededor de 100 veces por segundo de tiempo de ejecución, pero la frecuencia exacta puede variar de un sistema a otro.

Esto se hace de una de las dos maneras. La mayoría de los sistemas operativos tipo UNIX proporcionan una llamada al sistema 'profil()', que registra un vector de memoria en el kernel, junto con un factor de escala que determina cuanto espacio ocupan las direcciones del programa en el vector. Los valores típicos de escala son de 2 a 8 bytes de espacio de direcciones para mapear en un solo lugar del vector. A cada paso del reloj del sistema (suponiendo que el programa perfilado se está ejecutando), el valor del contador del programa se examina y el lugar correspondiente del array de memoria se incrementa. Dado que esto se hace en el kernel, que tenía que interrumpir el proceso de todas formas para manejar la interrupción de reloj, se requiere muy poca carga adicional en el sistema.

Sin embargo, algunos sistemas operativos, especialmente Linux 2.0 (y anteriores), no proporcionan una llamada al sistema 'profil()'. En tales sistemas, se hacen arreglos para el kernel para lanza periódicamente una señal para el proceso (normalmente a través de 'setitimer()'), que luego realiza la misma operación de examinar el contador del programa y el aumento de un lugar del vector de memoria. Dado que este método requiere de una señal que se lanza al espacio de usuario cada vez que se toma una muestra, se utiliza mucho más espacio que en los perfiles basado en el kernel. Además, debido al retraso añadido para lanzar la señal, este método también es menos preciso.

Un rutina de inicio especial asigna memoria para el histograma y, o bien llama 'profil()' o crea un manejador de la señal de reloj. Esta rutina ('monstartup') se puede invocar de varias maneras. En los sistemas Linux, un archivo de inicio especial para perfiles 'gcrt0.o', que invoca 'monstartup' antes que 'main', se utiliza en lugar del predeterminado 'crt0.o'. El uso de este archivo de inicio especial es uno de los efectos de la utilización de 'gcc ... -pg' como enlazador. En los sistemas SPARC, no se usan archivos especiales de inicio. En su lugar, la rutina 'mcount', cuando se invoca por primera vez (por lo general cuando se llama a 'main'), llama a 'monstartup'.

Si se utilizó la opción '-a' del compilador, se habilita también el recuento de bloques-básicos. Cada archivo objeto entonces se compila

con un vector estático de contadores, inicialmente cero. En el código ejecutable, cada vez que un bloque-básico comienza (por ejemplo, cuando aparece una sentencia 'if'), una instrucción adicional se introduce para incrementar la cuenta correspondiente del vector. En tiempo de compilación, un vector emparejado se construyó que registra la dirección de comienzo de cada bloque-básico. En conjunto, los dos vectores registran la dirección de inicio de cada bloque-básico, junto con el número de veces que se ejecutó.

La biblioteca de perfilado también incluye una función ('mcleanup') que normalmente se registra utilizando 'atexit()' para ser llamada cuando se cierra el programa, y es responsable de escribir el fichero 'gmon.out'. El perfilado se deshabilita, varias cabeceras se emiten, y se escribe el histograma, seguido por el grafo de llamadas El perfil se apaga, se emiten distintas cabeceras, y el histograma por escrito, seguido de los arcos del grafo de llamadas y los contadores de bloques-básicos

La salida de gprof no da ninguna indicación de las partes de tu programa que están limitadas por E/S ó intercambio de ancho de banda. Esto se debe a que las muestras del contador de programa se toman a intervalos fijos de tiempo de ejecución del programa. Por lo tanto, las mediciones de tiempo en la salida de gprof no dicen nada sobre el tiempo en que tu programa no se estaba ejecutando. Por ejemplo, una parte del programa que crea tanta cantidad de datos que no caben todos en la memoria física de una sola vez, puede funcionar muy lentamente debido a la hiper paginación, pero gprof dirá que utiliza muy poco tiempo. Por otro lado, el muestreo por el tiempo de ejecución tiene la ventaja de que la cantidad de carga debido a otros usuarios no afectará directamente a la salida que se obtiene.

1.9.2. Formato del fichero de datos del perfilado

El viejo formato de archivo derivado de BSD utilizado para los datos de perfilado no contiene una 'cookie' mágica que permite comprobar si un archivo de datos es realmente un archivo de gprof. Además, no proporciona un número de versión, haciendo así interpretar los cambios en el formato del archivo casi imposible. GNU gprof utiliza un nuevo formato de archivo que ofrece estas características. Por razones de compatibilidad, GNU gprof sigue soportando el viejo formato derivado de BSD, pero no todas las características se soportan en él. Por ejemplo, el recuento de la ejecución de bloques-básicos no se puede satisfacer pro el formato de archivo antiguo.

El nuevo formato de archivo se define en el fichero de cabecera 'gmon_out.h'. Se compone de una cabecera que contiene la 'cookie'

mágica y un número de versión, así como algunos bytes de reserva disponibles para futuras ampliaciones. Todos los datos en un archivo de datos de perfilado se encuentran en el formato nativo del objetivo al que se le está haciendo el perfilado. GNU gprof se adapta automáticamente al orden de bytes que se está usando.

En el nuevo formato de archivo, el encabezado es seguido por una secuencia de registros. En la actualidad, hay tres diferentes tipos de registros: Los registros de histograma, los registros de arcos del grafo de llamadas, y registros de contadores de ejecución de bloques básicos. Cada archivo puede contener cualquier número de cada tipo de registro. Al leer un archivo, GNU gprof se asegurará de que los registros del mismo tipo sean compatibles entre sí y calcular la unión de todos los registros. Por ejemplo, para el recuento de ejecuciones de bloques-básicos, la unión es simplemente la suma de todos los contadores de ejecución de cada bloque-básico.

Registro de histogramas

Los registros del histograma constan de una cabecera, seguida por un vector de contenedores. La cabecera contiene el intervalo del segmento de texto que abarca el histograma, el tamaño del histograma en bytes (a diferencia de en el antiguo formato BSD, este no incluye el tamaño de la cabecera), la tasa del reloj de perfilado, y la dimensión física que cuentan los contenedores representado después de haber sido escalados por el ritmo del reloj de perfilado. La dimensión física se especifica en dos partes: un nombre largo de hasta 15 caracteres y una abreviatura de un sólo carácter. Por ejemplo, un histograma que representa en tiempo real, tendría que especificar el nombre completo como 'segundos' y la abreviatura como 's'. Esta función es útil para las arquitecturas que soportan monitorización de rendimiento por 'hardware' (que, afortunadamente, se está convirtiendo en cada vez más común). Por ejemplo, en DEC OSF/1, el comando 'uprofile' se puede utilizar para producir un histograma de, por ejemplo, fallos de memoria caché. En este caso, la dimensión en la cabecera del histograma puede ser configurada en 'i-fallos de caché' y la abreviatura puede ser configurado como '1' (porque es simplemente un recuento, no una dimensión física). Además, la tasa de perfilado tendría que ser colocada a 1 en este caso.

Los contenedores de histograma son números de 16 bits y cada contenedor representa la misma cantidad de espacio de texto. Por ejemplo, si el segmento de texto es de mil bytes de longitud y si hay diez contenedores en el histograma, cada contenedor representa cien bytes.

Registro del grafo de llamadas

Los registros del grafo de llamada tienen un formato que es idéntico al usado en el formato de archivo derivado de BSD. Se compone de un arco del grafo de llamadas y un contador que indica el número de veces que se ha atravesado el arco durante la ejecución del programa. Los arcos se especifican como una pareja de direcciones: la primera debe estar dentro de la función que llama y la segunda debe estar dentro de la función destinataria de la llamada. Al realizar el perfilado a nivel de funciones, estas direcciones pueden apuntar a cualquier lugar dentro de la respectiva función. Sin embargo, cuando el perfilado es a nivel de líneas, es mejor si las direcciones están lo más cercanas como sea posible al lugar de la llamada ('call-site') o al lugar de entrada ('entry-point'). Esto asegurará que el grafo de llamadas a nivel de línea puede identificar exactamente qué línea de código fuente realizó la llamada a la función.

Registro de contadores de ejecución de bloques-básicos

Los registros de contadores de ejecución de bloques-básicos consisten en una cabecera seguida de una secuencia de parejas dirección/contador. La cabecera se limita a especificar la longitud de la secuencia. En una pareja dirección/contador, la dirección identifica a un bloque-básico y el contador especifica el número de veces que el bloque-básico se ejecutó. Cualquier dirección dentro del bloque-básico se pueden utilizar para identificarlo.

1.9.3. Funcionamiento interno de gprof

Como la mayoría de los programas, gprof comienza el procesamiento de sus propias opciones. Durante esta etapa, es posible que se construya la lista de 'symspecs' ('sym_ids.c:sym_id_add'), si se especifican las opciones que utilizan 'symspecs'. gprof mantiene una simple lista enlazada de 'symspecs', que finalmente se convertirán en una tabla de 12 símbolos, organizados en seis parejas incluir/excluir - una pareja para el perfil plano (INCL_FLAT / EXCL_FLAT), otra para los arcos del grafo de llamadas (INCL_ARCS / EXCL_ARCS), otra para imprimir en el grafo de llamadas (INCL_GRAPH / EXCL_GRAPH), otra para la propagación del tiempo en el grafo de llamadas (INCL_TIME / EXCL_TIME), otra para el listado del fuente con anotaciones (INCL_ANNO / EXCL_ANNO), y otra para el listados de contadores de ejecución (INCL_EXEC / EXCL_EXEC).

Después de procesar las opciones, gprof termina de construir la lista de 'symspec' añadiendo todos los 'symspecs' en 'default_excluded_list' para las lista de exclusión EXCL_TIME y EXCL_GRAPH, y si el perfilado línea-a-línea se especifica EXCL_FLAT, también. Estas exclusiones por defecto no se agregan a EXCL_ANNO, EXCL_ARCS y EXCL_EXEC.

A continuación, la biblioteca BFD se llama para abrir el archivo objeto, comprobar que realmente es un archivo objeto, y leer su tabla de símbolos ('core.c: core_init'), utilizando 'bfd_canonicalize_symtab' después de asignar un tamaño apropiado de memoria ('malloc') para el array de símbolos. En este punto, se lee el mapeado de las funciones (si la opción '–file-ordering' se ha especificado), y se lee el espacio de texto básico de la memoria (si se dio la opción '-c').

La tabla de símbolos propia de gprof, un vector de estructuras 'Sym', se construye ahora. Esto se hace de una de dos maneras, por una de las dos rutinas, dependiendo de si se ha habilitado un perfilado línea-a-línea (opción '-l'). Para el perfilado normal, la tabla de símbolo BFD canónica se escanea. Para el perfilado línea-a-línea, se examina cada espacio de direcciones de texto, y se crea una nueva entrada en la tabla de símbolos cada vez que cambia el número de línea. En cualquier caso, se hacen dos pasadas a través de la tabla de símbolos - una para contabilizar el tamaño necesario para la tabla de símbolos, y la otra para leer realmente los símbolos. Entre pasada y pasada, se crea un vector de tipo 'Sym' con la longitud adecuada. Por último, se llama a 'symtab.c:symtab_finalize' para ordenar la tabla de símbolos y eliminar las entradas duplicadas (entradas con la misma dirección de memoria).

La tabla de símbolos debe ser un vector contiguo por dos razones. En primer lugar, la función de biblioteca 'qsort' (que ordena el vector) se utilizará para ordenar la tabla de símbolos. Además, la rutina de búsqueda de símbolos ('symtab.c: sym_lookup'), que encuentra símbolos basándose en la dirección de memoria, utiliza un algoritmo de búsqueda binaria que requiere que la tabla de símbolos sea un vector ordenado. Los símbolos de funciones se indican con una bandera 'is_func' (es función). Los símbolos para el número de línea no tienen configurada ningún indicador especial. Además, un símbolo puede tener una bandera 'is_static' para indicar que es un símbolo local.

Con la tabla de símbolos leída, los 'symspecs' ahora se puede traducir en 'Syms' ('sym_ids.c: sym_id_parse). Recuerda que un simple 'symspec' puede identificar varios símbolos. Se crea un vector de tablas de símbolos ('syms'), cada entrada de la cual es una tabla de símbolos de 'Syms' que serán incluidos o excluidos de una lista particular. La tabla de símbolos maestros y los 'symspecs' se examinan por bucles anidados, y todos los símbolos que concuerdan con un

'smspec' se insertan en la tabla 'syms' apropiado. Esto se hace dos veces, una para contar el tamaño requerido para cada tabla de símbolos, y otra vez para construir las tablas que fueron asignadas entre una pasada y la otra. A partir de ahora, para determinar si un símbolo está en una lista de 'symspec' de inclusión o de exclusión, gprof simplemente utiliza su rutina estándar búsqueda de símbolos en la tabla correspondiente del vector de 'syms'.

Ahora el archivo(s) de datos de perfilado son leídos ('gmon_io.c: gmon_out_read), primero para comprobar el estilo nuevo del encabezado de 'gmon.out', y después, asumiendo que es un 'gmon.out' al viejo estilo BSD si falló el test de la 'cookie' mágica.

Los registros de histograma al nuevo estilo se leen por 'hist.c:hist_read_rec'. Para el primer registro de histograma, se asigna un vector de memoria para almacenar todos los contenedores, y después leerlos. Cuando se leen varios archivos de datos de perfilado (o un archivo con varios registros de histograma), la dirección de inicio, la dirección de terminación, el número de contenedores y la frecuencia de muestreo deben coincidir entre los distintos histogramas, o ocurrirá un error fatal. Si todo coincide, sólo suma los histogramas adicionales al vector existente en la memoria.

Tal como se lee cada registro del grafo de llamadas ('call_graph.c:cg_read_rec'), las direcciones de padres e hijos son comprobadas con las entradas en la tabla de símbolos, y un arco del grafo de llamadas se crea por 'cg_arcs.c:arc_add', a menos que el arco falle una comprobación de 'symspec' contra INCL_ARCS/EXCL_ARCS.. Tal como se añade cada arco, se mantiene una lista enlazada con los arcos hijos de los padres, y de los arcos padres de los hijos. Ambos el contador de llamadas al hijo y el contador de llamadas recibidas se incrementan por el contador del registro de llamadas.

Los bloques-básicos se leen ('basic_blocks.c:bb_read_rec'), pero sólo si se selecciona el perfilado línea-a-línea. Cada dirección del bloque-básico se comprueba si corresponde con una línea de símbolo de la tabla de símbolos, y una entrada hecha en el vector de símbolos bb_addr y bb_calls. Una vez más, si varios registros de bloques-básicos se presentan para la misma dirección, el número de llamadas se acumulan.

Un fichero 'gmon.sum' se guarda, si así lo solicita ('gmon_io.c:gmon_out_write').

Si se presentaron histogramas en los archivos de datos, se asignan a los símbolos ('hist.c:hist_assign_samples') iterando por todas los contenedores de la muestra y asignándolos a los símbolos. Como la tabla de símbolos está ordenada en orden ascendente de dirección de

memoria, podemos simplemente seguir a través de la tabla de símbolos hasta que hagamos nuestra pasada sobre el contenedor de muestra. Este paso incluye una comprobación contra INCL_FLAT/EXCL_FLAT. Dependiendo del factor de escala del histograma, un contenedor de muestra puede abarcar varios símbolos, en cuyo caso se le asigna una fracción del contador de muestra a cada símbolo, proporcional al grado de superposición. Este efecto es raro para los perfilados normales, pero la superposición es más común durante el perfilado línea-a-línea, y puede causar cada una de dos líneas adyacentes reciban la mitad de un éxito, por ejemplo.

Si se presentan datos del grafo de llamadas, se llama a 'cg_arcs.c:cg_assemble'. En primer lugar, si se especificó la opción '-c', una rutina dependiente de la máquina ('find_call') explora a través de código máquina, cada símbolo, en busca de las instrucciones de llamada a la subrutina, y los agrega al grafo de llamadas con su contador de llamadas a cero. Una ordenación topológica se realiza en profundidad para numerar todos los símbolos ('cg_dfn.c:cg_dfn'), para que los hijos se numeren siempre menos que sus padres, y luego hacer un vector de punteros a la tabla de símbolos y ordenarlos en orden numérico, que es el orden inverso al topológico (los hijos antes que sus padres). Los ciclos también se detectan en este momento, todos los miembros que tienen asignado el mismo número topológico. Dos pasadas se hacen ahora a través de este vector ordenado de punteros a símbolos. La primera pasada, desde el final al principio (de padres a hijos), calcula la fracción de tiempo del hijo que se propaga para cada padre e imprime un bandera. La bandera impresa refleja el manejo de 'symspec' de INCL_GRAPH/EXCL_GRAPH, con el padre incluido o excluido (imprimir o no imprimir) apropiadamente propagado a sus hijos, a menos que ellos aparezcan explícitamente en INCL_GRAPH o EXCL_GRAPH. Una segunda pasada, de principio a fin (a los hijos a los padres) en realidad propaga los tiempos a lo largo del grafo de llamadas, objeto de un control contra INCL_TIME/EXCL_TIME. Con la bandera de impresión, las fracciones y los tiempos ahora guardados en las estructuras de símbolos, el vector ordenado topológicamente está ahora descartado, y un nuevo vector de punteros se monta, esta vez ordenados por tiempo propagado.

Finalmente, imprime las diferentes salidas que el usuario solicita, que ahora es bastante sencillo. El grafo de llamadas ('cg_print.c:cg_print') y el perfil plano ('hist.c:hist_print') son regurgitaciones de los valores ya calculados. El listado del fuente con anotaciones ('basic_blocks.c:print_annotated_so usa información de los bloques-básicos, si están presentes, para etiquetar cada línea de código con el número de llamadas, de lo contrario sólo cuenta la llamada a la función.

El código de ordenación de funciones está marginalmente bien documentado en el propio código fuente ('cg_print.c'). Básicamente, las funciones con el mayor uso y con más padres se posicionan primeras, seguidas por el resto de funciones de mayor uso, seguidas por las funciones de uso más bajo, seguidas finalmente por las funciones no utilizadas.

1.9.4. Depurar gprof

Si gprof se compila con el depurado habilitado, la opción '-d' dispara la salida de depuración (a la salida estándar) que puede ser útil para entender su funcionamiento. El número de depurado especificado se interpreta como una suma de las siguientes opciones:

2 - Orden topológico Monitoriza la numeración en profundidad de símbolos durante el análisis del grafo de llamadas

4 - Ciclos Muestra símbolos como si fueran identificado como cabeceras de ciclo

16 - Concordancia Tal como el arco del grafo de llamadas se lee, muestra cada arco y cómo se actualiza el total de llamadas de cada función que concuerda.

32 - Ordenación del arco del grafo de llamadas Detalla el orden individual de los padres/hijos en cada entrada del grafo de llamadas

64 - Leer el histograma y los registro del grafo de llamadas Muestra los rangos de direcciones de los histogramas tal como se leen, y cada arco del grafo de llamadas.

128 - Tabla de símbolos Lectura, clasificación y ordenación de la tabla de símbolos del fichero objeto. Para el perfilado línea-a-línea (opción '-l'), también muestra los números de las líneas que se asignan a las direcciones de memoria.

256 - Grafo de llamadas estático Traza el funcionamiento de la opción '-c'

512 - Búsquedas en tabla de símbolos y la tabla de arcos Detalla el funcionamiento de la búsqueda de rutinas

1024 - Propagación del grafo de llamadas Muestra cómo se propagan las funciones de tiempo a lo largo del grafo de llamadas

2048 - Bloques-básicos Muestra los registro de los bloques-básicos tal como se leen del fichero de datos de perfilado (sólo valioso en conjunto con la opción '-l')

4096 - Symspecs Muestra el la operación de concordancia entre los símbolos y los symspec

8192 - Fuente anotado Sigue la pista al funcionamiento de la opción ‘-A’

1.10. GNU Free Documentation License

GNU Free Documentation License

Version 1.1, March 2000

Como este documento sigue la misma licencia que el proyecto (apartado 2, página 45), sólo hay una versión del mismo.

CAPÍTULO 2

GNU Free Documentation License

Este documento está protegido bajo la licencia de documentación libre *Free Documentacion License* del Proyecto GNU, que a continuación se incluye, toda persona que lo desee está autorizado a usar, copiar y modificar este documento según los puntos establecidos en la «Licencia FDL».

Este manual se distribuye sin garantía alguna, los autores no se hacen responsables de cualquier tipo de daño mental, emocional o de cualquier otro tipo que produzca la lectura o cualquier otro uso de este manual, usalo a tu propio riesgo.

2.1. GNU Free Documentation License

Version 1.2, November 2002

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to

assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

2.2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here

XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2.3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 2.4.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

2.4. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

2.5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2.3 and 2.4 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

Ñ. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

2.6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 2.5 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number.

Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

2.7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

2.8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 2.4 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

2.9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 2.5. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 2.5) to Preserve its Title (section 2.2) will typically require changing the actual title.

2.10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

2.11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.