Profiling

GPROF && Perf Tutorial

Introducción

Cuando escribimos código, generalmente tenemos cierta "intuición" de cuán eficiente es (tiempo de ejecución y/o uso de memoria) Eso podría estar respaldado por un análisis de complejidad de O grande de los algoritmos que usamos (por ejemplo, O(logN), O(N), etc). Desafortunadamente, eso no significa que estemos en lo correcto, por lo que a menudo queremos una validación experimental de qué tan rápido se ejecuta (o cuánta memoria usa) en la práctica. Las herramientas para analizar el tiempo de ejecución del programa/uso de memoria se llaman generadores de perfiles. Cómo funcionan los perfiladores de código (tiempo). Los generadores de perfiles de código a menudo se usan para analizar no solo cuánto tiempo tarda en ejecutarse un programa (podemos obtenerlo de herramientas a nivel de shell como /usr/bin/time), sino también cuánto tiempo tarda en ejecutarse cada función o método (tiempo de CPU). Dos técnicas principales utilizadas por los perfiladores: inyección de código, muestreo.

GNU GCC Profiling (gprof)

Traducción del tutorial de Himanshu Arora (2012) y adaptación por Javier JORGE (2023)
https://www.thegeekstuff.com/2012/08/gprof-tutorial/
mas info en
https://linuxhint.com/gprof-linux-command/
https://blog.mbedded.ninja/programming/compilers/gcc/gcc-profiling/#gprof2dot
https://dev.to/etcwilde/perf---perfect-profiling-of-cc-on-linux-of
https://github.com/google/benchmark

Introducción

Usar la herramienta gprof no es nada complejo. Solo necesita hacer lo siguiente:

- Habilitar “profiling” durante la compilación
- Ejecutar el código del programa para producir los datos de perfil
- Ejecute la herramienta gprof en el archivo de datos de generación de perfiles (generado en el paso anterior).

El último paso anterior produce un archivo de análisis que está en forma legible por humanos.

Este archivo contiene un par de tablas (perfil plano y gráfico de llamadas) además de otra información. Mientras que el perfil plano brinda una descripción general de la información de tiempo de las funciones, como el consumo de tiempo para la ejecución de una función en particular, cuántas veces se llamó, etc. Por otro lado, el gráfico de llamadas se enfoca en cada función como las funciones a través de las cuales un determinado se llamó a la función, qué funciones se llamaron desde dentro de esta función en particular, etc. De esta manera, también se puede tener una idea del tiempo de ejecución empleado en las subrutinas.

Tratemos de entender los tres pasos enumerados anteriormente a través de un ejemplo práctico.

El siguiente código de prueba se utilizará en todo el artículo:

```
//test_gprof.c
#include<stdio.h>

void new_func1(void);

void func1(void)
{
    printf("\n Inside func1 \n");
    int i = 0;

    for(;i<0xffffffff;i++);
    new_func1();

    return;
}

static void func2(void)
{
    printf("\n Inside func2 \n");
    int i = 0;

    for(;i<0xffffffaa;i++);
    return;
}

int main(void)
{
    printf("\n Inside main()\n");
    int i = 0;
```

```
    for(;i<0xffffff;i++);
    func1();
    func2();

    return 0;
}
//test_gprof_new.c
#include<stdio.h>

void new_func1(void)
{
    printf("\n Inside new_func1()\n");
    int i = 0;

    for(;i<0xfffffee;i++);

    return;
}
```

Tenga en cuenta que los bucles 'for' dentro de las funciones están ahí para consumir algo de tiempo de ejecución.

Paso 1: creación de perfiles habilitada durante la compilación

En este primer paso, debemos asegurarnos de que la generación de perfiles esté habilitada cuando se complete la compilación del código. Esto es posible al agregar la opción '-pg' en el paso de compilación.

Del man de gcc:

> -pg : Generate extra code to write profile information suitable for the analysis program gprof. You must use this option when compiling the source files you want data about, and you must also use it when linking.

Entonces, compilemos nuestro código con la opción '-pg':

```
$ gcc -Wall -pg test_gprof.c test_gprof_new.c -o test_gprof
```

Paso 2: Ejecutar el código

En el segundo paso, se ejecuta el archivo binario producido como resultado del paso 1 (arriba)

para que se pueda generar la información de perfiles.

```
$ ls
test_gprof  test_gprof.c  test_gprof_new.c

$ ./test_gprof
 Inside main()
 Inside func1
 Inside new_func1()
 Inside func2

$ ls
gmon.out  test_gprof  test_gprof.c  test_gprof_new.c
```

Entonces vemos que cuando se ejecuta el binario, se genera un nuevo archivo 'gmon.out' en el directorio de trabajo actual.

Tenga en cuenta que durante la ejecución, si el programa cambia el directorio de trabajo actual (usando chdir), se generará gmon.out en el nuevo directorio de trabajo actual. Además, su programa debe tener permisos suficientes para que gmon.out se cree en el directorio de trabajo actual.

Paso 3: Ejecute la herramienta gprof

En este paso, la herramienta gprof se ejecuta con el nombre del ejecutable y el 'gmon.out' generado anteriormente como argumento. Esto produce un archivo de análisis que contiene toda la información de perfil deseada.

```
$  gprof test_gprof gmon.out > analysis.txt
$ ls
analysis.txt  gmon.out  test_gprof  test_gprof.c  test_gprof_new.c
```

Entonces vemos que se generó un archivo llamado 'analysis.txt'.

Comprensión de la información de perfil

Cómo se produjo anteriormente, toda la información de perfil ahora está presente en 'analysis.txt'. Echemos un vistazo a este archivo de texto:

```
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative self              self   total
time seconds    seconds calls s/call s/call name
```

```
33.86 15.52     15.52    1    15.52  15.52  func2
33.82 31.02     15.50    1    15.50  15.50  new_func1
33.29 46.27     15.26    1    15.26  30.75  func1
0.07  46.30     0.03                   main
```

% el porcentaje del tiempo total de funcionamiento del programa de tiempo utilizado para esta función.

cumulative una suma corriente del número de segundos contabilizados, segundos por esta función y las enumeradas anteriormente.

self el número de segundos contabilizados por este los segundos funcionan solos. Este es el tipo principal para este listado.

calls al número de veces que se invocó esta función, si esta función está perfilada, de lo contrario en blanco.

self el número promedio de milisegundos gastados en este ms/función de llamada por llamada, si esta función está perfilada, de lo contrario en blanco.

total el número promedio de milisegundos gastados en este función ms/call y sus descendientes por llamada, si esto la función está perfilada, de lo contrario, en blanco.

name el nombre de la función. Este es el tipo menor para este listado. El índice muestra la ubicación de la función en la lista de gprof. Si el índice es entre paréntesis muestra dónde aparecería en el listado de gprof si fuera a ser impreso.

Call graph (explanation follows)

granularity: each sample hit covers 2 byte(s) for 0.02% of 46.30 seconds

index % time self children called name

```
[1]   100.0  0.03  46.27          main [1]
           15.26 15.50    1/1      func1 [2]
           15.52 0.00     1/1      func2 [3]
-----------------------------------------------
           15.26 15.50    1/1      main [1]
[2]   66.4   15.26 15.50    1     func1 [2]
           15.50 0.00     1/1      new_func1 [4]
-----------------------------------------------
           15.52 0.00     1/1      main [1]
[3]   33.5   15.52 0.00     1     func2 [3]
```

```
-----------------------------------------------
                15.50 0.00      1/1       func1 [2]
[4] 33.5        15.50 0.00      1         new_func1 [4]
-----------------------------------------------
```

Esta tabla describe el árbol de llamadas del programa y fue ordenada por la cantidad total de tiempo empleado en cada función y sus hijos.

Cada entrada en esta tabla consta de varias líneas. La línea con el número de índice en el margen izquierdo enumera la función actual. Las líneas arriba enumeran las funciones que llamaron a esta función, y las líneas debajo enumeran las funciones a las que llama.
Esta línea enumera:
index Un número único dado a cada elemento de la tabla. Los números de índice se ordenan numéricamente. El número de índice está impreso al lado de cada nombre de función para que es más fácil buscar dónde está la función en la tabla.

% time Este es el porcentaje del tiempo 'total' que se dedicó en esta función y sus hijos. Tenga en cuenta que debido a diferentes puntos de vista, funciones excluidas por opciones, etc. estos números NO sumarán 100%.

self Esta es la cantidad total de tiempo empleado en esta función.

children Esta es la cantidad total de tiempo propagado en esta función por sus hijos.

calls Este es el número de veces que se llamó a la función. Si la función se llama a sí misma recursivamente, el número solo incluye llamadas no recursivas, y es seguido por un `+' y el número de llamadas recursivas.

name El nombre de la función actual. El número de índice es impreso después de él. Si la función es miembro de un ciclo, el número de ciclo se imprime entre el el nombre de la función y el número de índice.

Para los padres de la función, los campos tienen los siguientes significados:

self Esta es la cantidad de tiempo que se propagó directamente de la función a este padre.

childs Esta es la cantidad de tiempo que se propagó desde los hijos de la función en este padre.

calls Este es el número de veces que este padre llamó al función `/' el número total de veces que la función fue llamado Las llamadas recursivas a la función no son incluido en el número después de `/'.

name Este es el nombre del padre. El índice de los padres el número se imprime después. Si el padre es un miembro de un ciclo, el número de ciclo se imprime entre el nombre y el número de índice.

Si no se pueden determinar los padres de la función, la palabra `' está impreso en el campo `nombre', y todos los demás campos están en blanco.

Para los hijos de la función, los campos tienen los siguientes significados:

self Esta es la cantidad de tiempo que se propagó directamente del hijo a la función.

hijos Esta es la cantidad de tiempo que se propagó desde el los hijos del hijo a la función.

llamada Este es el número de veces que la función llamada este hijo `/' el número total de veces que el hijo fue llamado Las llamadas recursivas del hijo no son aparece en el número después de `/'.

nombre Este es el nombre del hijo. el índice del hijo el número se imprime después. Si el hijo es miembro de un ciclo, el número de ciclo se imprime entre el nombre y el número de índice.

Si hay ciclos (círculos) en el gráfico de llamadas, hay un entrada para el ciclo como un todo. Esta entrada muestra quién llamó al ciclo (como padres) y los miembros del ciclo (como hijos). La entrada de llamadas recursivas `+' muestra el número de llamadas de función que eran internos al ciclo, y la entrada de llamadas para cada miembro muestra, para ese miembro, ¿cuántas veces fue llamado de otros miembros de el ciclo.
Index by function name

[2] func1 [1] main
[3] func2 [4] new_func1

So (as already discussed) we see that this file is broadly divided into two parts :

1. Flat profile
2. Call graph

The individual columns for the (flat profile as well as call graph) are very well explained in the output itself.

Customize gprof output using flags

There are various flags available to customize the output of the gprof tool. Some of them are discussed below:

1. Suprima la impresión de funciones declaradas estáticamente (privadas) usando -a

$ gprof -a test_gprof gmon.out > analysis.txt

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
67.15	30.77	30.77	2	15.39	23.14	func1
33.82	46.27	15.50	1	15.50	15.50	new_func1
0.07	46.30	0.03				main

...
...
...

Call graph (explanation follows)

granularity: each sample hit covers 2 byte(s) for 0.02% of 46.30 seconds

index	%time	self	children	called	name
[1]	100.0	0.03	46.27		main [1]
		30.77	15.50	2/2	func1 [2]

		30.77	15.50	2/2	main [1]
[2]	99.9	30.77	15.50	2	func1 [2]
		15.50	0.00	1/1	new_func1 [3]

		15.50	0.00	1/1	func1 [2]
[3]	33.5	15.50	0.00	1	new_func1 [3]

...
...
...

Entonces vemos que no hay información relacionada con func2 (que se define como estática)

2. Elimine los textos detallados usando -b

$ gprof -b test_gprof gmon.out > analysis.txt

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
33.86	15.52	15.52	1	15.52	15.52	func2
33.82	31.02	15.50	1	15.50	15.50	new_func1
33.29	46.27	15.26	1	15.26	30.75	func1
0.07	46.30	0.03				main

Call graph

granularity: each sample hit covers 2 byte(s) for 0.02% of 46.30 seconds

```
index % time    self  children    called     name

[1]    100.0    0.03   46.27                 main [1]
                15.26   15.50       1/1       func1 [2]
                15.52    0.00       1/1       func2 [3]
-----------------------------------------------
                15.26   15.50       1/1       main [1]
[2]     66.4    15.26   15.50       1         func1 [2]
                15.50    0.00       1/1       new_func1 [4]
-----------------------------------------------
                15.52    0.00       1/1       main [1]
[3]     33.5    15.52    0.00       1         func2 [3]
-----------------------------------------------
                15.50    0.00       1/1       func1 [2]
[4]     33.5    15.50    0.00       1         new_func1 [4]
-----------------------------------------------
```

Index by function name

[2] func1 [1] main
[3] func2 [4] new_func1

3. Imprima solo perfil plano usando -p

$ gprof -p -b test_gprof gmon.out > analysis.txt

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
33.86	15.52	15.52	1	15.52	15.52	func2
33.82	31.02	15.50	1	15.50	15.50	new_func1
33.29	46.27	15.26	1	15.26	30.75	func1
0.07	46.30	0.03				main

4. Imprimir información relacionada con funciones específicas en perfil plano

$ gprof -pfunc1 -b test_gprof gmon.out > analysis.txt

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
103.20	15.26	15.26	1	15.26	15.26	func1

Genere un gráfico

gprof2dot es una herramienta que puede crear una visualización de la salida de gprof.

instalar gprof2dot:
$ pip instalar gprof2dot

instalar graphviz (que es necesario si va a hacer gráficos de "puntos" como el siguiente):
$sudo apt install graphviz

Profiling con linux perf

Introducción

Perf es una pequeña herramienta que acabo de encontrar para crear perfiles de programas. Perf utiliza perfiles estadísticos, donde sondea el programa y ve qué función está funcionando.

Esto es menos preciso, pero tiene menos impacto en el rendimiento que algo como Callgrind, que rastrea cada llamada. Los resultados siguen siendo razonablemente precisos, e incluso con menos muestras, mostrará qué funciones están tomando mucho tiempo, incluso si pierde funciones que son muy rápidas (que probablemente no sean las que está buscando al perfilar de todos modos).
https://dev.to/etcwilde/perf---perfect-profiling-of-cc-on-linux-of

instalación

sudo apt install linux-tools-common
sudo apt install linux-tools-5.19.0-35-generic

Ejecución

sudo perf record ./examples/test_gprof
sudo perf report