PRACTICO #1 RENDIMIENTO

Gaston Marcelo Segura

gastonsegura2908@mi.unc.edu.ar

42935935

Universidad Nacional de Córdoba

Sistemas de Computación

Javier Alejandro Jorge

25/03/2024

Resolución de consignas

Las herramientas para analizar el tiempo de ejecución del programa/uso de memoria se llaman generadores de perfiles. Dos técnicas principales utilizadas por los perfiladores: inyección de código(GPROF), muestreo/sampling(PERF).

GPROF

Para usarla hay que realizar 3 PASOS:

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

PASO 1: Habilitar "profiling" durante la compilación

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.

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

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

```
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ gcc -Wall -pg test_gprof.c test_gprof_new.c -o test_gprof
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$
```

Acá está lo que hace cada parte:

- `gcc`: Es el compilador de GNU para C y C++. Este comando se usa para compilar código fuente en C y C++.
- `-Wall`: Esta opción habilita todas las advertencias sobre construcciones que algunos usuarios consideran cuestionables, y que son fáciles de evitar.
- `-pg`: Esta opción instruye al compilador para insertar código adicional en el ejecutable para recopilar información sobre el tiempo de ejecución. Esta información se puede utilizar para el análisis de rendimiento con una herramienta como gprof.
- `test_gprof.c test_gprof_new.c`: Estos son los archivos de código fuente que se están compilando. En este caso, hay dos archivos de código fuente, `test_gprof.c` y `test_gprof_new.c`.

- `-o test_gprof`: La opción `-o` se utiliza para especificar el nombre del archivo de salida. En este caso, el archivo de salida (el ejecutable) se llamará `test_gprof`.

Por lo tanto, este comando compila los archivos `test_gprof.c` y `test_gprof_new.c` en un ejecutable llamado `test_gprof`, con todas las advertencias habilitadas y con la capacidad de realizar un análisis de rendimiento utilizando gprof.

PASO 2: Ejecutar el código del programa para producir los datos de perfil

Se ejecuta el archivo binario(test_gprof) producido como resultado del paso 1

```
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ ls
archivo1.c  archivo2.c  archivo3.c  Prueba  test_gprof  test_gprof.c  test_gprof_new.c
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ ./test_gprof

 Inside main()

 Inside func1

 Inside new_func1()

 Inside func2
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ ls
archivo1.c  archivo2.c  archivo3.c  gmon.out  Prueba  test_gprof  test_gprof.c  test_gprof_new.c
```

Entonces vemos que cuando se ejecuta el binario(al hacer ./test_gprof), se genera un nuevo archivo 'gmon.out' en el directorio de trabajo actual.

PASO 3: Ejecutar la herramienta gprof en el archivo de datos de generación de perfiles (generado en el paso anterior).

La herramienta gprof se ejecuta con el nombre del ejecutable(test_gprof) y el 'gmon.out' generado anteriormente como argumento. Esto produce un archivo de análisis que contiene toda la información de perfil deseada.

```
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ ls
archivo1.c  archivo2.c  archivo3.c  gmon.out  Prueba  test_gprof  test_gprof.c  test_gprof_new.c
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ gprof test_gprof gmon.out>analysis.txt
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ ls
analysis.txt  archivo1.c  archivo2.c  archivo3.c  gmon.out  Prueba  test_gprof  test_gprof.c  test_gprof_new.c
```

Vemos que se generó un archivo llamado 'analysis.txt'.

Ahora procedemos a abrir ese archivo analysis.txt, con el comando cat:

```
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ ls
analysis.txt  archivo1.c  archivo2.c  archivo3.c  gmon.out  Prueba  test_gprof  test_gprof.c  test_gprof_new.c
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ ./analysis.txt
bash: ./analysis.txt: Permiso denegado
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ cat analysis.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 34.93     12.19    12.19        1    12.19    23.66  func1
 32.87     23.66    11.47        1    11.47    11.47  new_func1
 32.09     34.86    11.20        1    11.20    11.20  func2
  0.11     34.90     0.04                             main

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
```

Entonces (como ya se discutió) vemos que **este archivo se divide en términos generales en dos partes:**

1. **Perfil plano(flat profile)**
2. **Gráfico de llamadas(Call graph)**

Las columnas individuales para (perfil plano y gráfico de llamadas) están muy bien explicadas en el resultado mismo.

1. **Flat Profile:** Muestra cuánto tiempo pasó tu programa en cada función y cuántas veces se llamó a esa función. Si simplemente quieres saber qué funciones consumen la mayoría de los ciclos, se indica de manera concisa aquí.

Acá está el significado de cada columna:

time: El porcentaje del tiempo total de ejecución del programa gastado en esta función.
acumulative seconds: El tiempo de ejecución acumulado en esta función y en las funciones que aparecen antes de ella en la lista.

self seconds: El tiempo total gastado en esta función.
calls: El número total de veces que se llamó a la función.
self s/call: El tiempo promedio gastado en esta función cada vez que se llamó, si se llamó al menos una vez.
total s/call: El tiempo promedio gastado en esta función y en las funciones que llama, si la función se llamó al menos una vez.
name: El nombre de la función.
Por ejemplo, en el perfil, la función func1 se llamó una vez, tomó 12.19 segundos para ejecutarse, y gastó un total de 23.66 segundos incluyendo las funciones que llamó. Esto representa el 34.93% del tiempo total de ejecución del program.a

2. **Call Graph:** Muestra, para cada función, qué funciones la llamaron, qué otras funciones llamó y cuántas veces. También hay una estimación de cuánto tiempo se pasó en las subrutinas de cada función[1]. Esto puede sugerir lugares donde podrías intentar eliminar llamadas a funciones que consumen mucho tiempo.

Acá está el significado de cada columna:

index: Un número único que identifica la función en el gráfico de llamadas.
time: El porcentaje del tiempo total de ejecución del programa gastado en esta función y sus hijos.
self: El tiempo total gastado en esta función.
children: El tiempo total gastado en las funciones llamadas por esta función (sus hijos).
called: El número de veces que se llamó a esta función y el número total de veces que se podría haber llamado.
name: El nombre de la función.
Por ejemplo, en el gráfico de llamadas, la función main (index [1]) llamó a las funciones func1 y func2. La función func1 a su vez llamó a new_func1.

PARA PERSONALIZAR LA SALIDA DE GPROF USANDO BANDERAS:

Hay varias flags disponibles para personalizar la salida de la herramienta gprof. Algunos de ellos se analizan a continuación:

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

```
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ gprof -a test_gprof gmon.out>analysis.txt
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ ls
analysis.txt  archivo1.c  archivo2.c  archivo3.c  gmon.out  Prueba  test_gprof  test_gprof.c  test_gprof_new.c
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ cat analysis.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 67.02     23.39    23.39        2    11.70    17.43  func1
 32.87     34.86    11.47        1    11.47    11.47  new_func1
  0.11     34.90     0.04                             main

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.
```

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

```
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ gprof -b test_gprof gmon.out>analysis.txt
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ ls
analysis.txt  archivo1.c  archivo2.c  archivo3.c  gmon.out  Prueba  test_gprof  test_gprof.c  test_gprof_new.c
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ cat analysis.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 34.93     12.19    12.19        1    12.19    23.66  func1
 32.87     23.66    11.47        1    11.47    11.47  new_func1
 32.09     34.86    11.20        1    11.20    11.20  func2
  0.11     34.90     0.04                             main


                     Call graph


granularity: each sample hit covers 4 byte(s) for 0.03% of 34.90 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.04   34.86                 main [1]
               12.19   11.47       1/1           func1 [2]
               11.20    0.00       1/1           func2 [4]
-----------------------------------------------
               12.19   11.47       1/1           main [1]
[2]     67.8   12.19   11.47       1         func1 [2]
               11.47    0.00       1/1           new_func1 [3]
-----------------------------------------------
               11.47    0.00       1/1           func1 [2]
[3]     32.9   11.47    0.00       1         new_func1 [3]
-----------------------------------------------
               11.20    0.00       1/1           main [1]
[4]     32.1   11.20    0.00       1         func2 [4]
-----------------------------------------------


Index by function name
```

3. Imprimir <u>solo perfil plano</u> usando -p

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

```
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ gprof -p -b test_gprof gmon.out>analysis.txt
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ ls
analysis.txt  archivo1.c  archivo2.c  archivo3.c  gmon.out  Prueba  test_gprof  test_gprof.c  test_gprof_new.c
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ cat analysis.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 34.93     12.19    12.19        1    12.19    23.66  func1
 32.87     23.66    11.47        1    11.47    11.47  new_func1
 32.09     34.86    11.20        1    11.20    11.20  func2
  0.11     34.90     0.04                             main
```

4. Imprimir información relacionada con funciones específicas en perfil plano

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

```
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ gprof -pfunc1 -b test_gprof gmon.out>analysis.txt
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ ls
analysis.txt  archivo1.c  archivo2.c  archivo3.c  gmon.out  Prueba  test_gprof  test_gprof.c  test_gprof_new.c
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ cat analysis.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.00     12.19    12.19        1    12.19    12.19  func1
```

Genere un gráfico

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

Para instalar gprof2dot:

```
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ pip install gprof2dot
Defaulting to user installation because normal site-packages is not writeable
Collecting gprof2dot
  Downloading gprof2dot-2022.7.29-py2.py3-none-any.whl (34 kB)
Installing collected packages: gprof2dot
  WARNING: The script gprof2dot is installed in '/home/gaston/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed gprof2dot-2022.7.29
```

Para instalar graphviz :(que es necesario si vamos a hacer gráficos de "puntos")

```
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ sudo apt install graphviz
[sudo] contraseña para gaston:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
```

PROFILING CON LINUX **PERF**

Perf es herramienta 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. Muestra qué funciones están tomando mucho tiempo.

Instalación

```
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ sudo apt install linux-tools-common
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
```

```
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ sudo apt install linux-tools-5.15.0-82-generic
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Los paquetes indicados a continuación se instalaron de forma automática y ya no son necesarios.
```

Ejecución

```
gaston@gaston-Lenovo-ideapad-320-14IAP:~/Documentos/nuevaprueba1$ sudo perf record ./test_gprof
[sudo] contraseña para gaston:

 Inside main()

 Inside func1

 Inside new_func1()

 Inside func2
[ perf record: Woken up 23 times to write data ]
[ perf record: Captured and wrote 5,539 MB perf.data (144519 samples) ]
```

Observamos que perf se despertó 23 veces para escribir los datos de los eventos que estaba grabando, capturó los datos de los eventos y los escribió en un archivo llamado perf.data. El tamaño de los datos es de 5,539 MB y se recogieron 144519 muestras.
Al poner en la terminal los comandos: sudo perf report , podemos obtener información mas detallada: