Usando o GNU Profiler

Pedro Garcia

http://www.sawp.com.br

22 de Janeiro de 2010

Assembly Working Party

Laboratório de Cálculos Científicos, Instituto de Física, Universidade de Brasília, Brasil

Aplicação

- Usado para determinar performance.
- Fornece o "perfil" das funções do programa.
- Motivação:
  - Analisar a diferença de performance entre diferentes assemblys gerados (quando estudarmos otimização).
  - Aprender a utilizar o profiler quando estivermos programando direto em ASM.

Funcionamento

- Recebe o programa (devidamente compilado) como parâmetro e retorna um relatório com as seguintes informações:
  1. Um resumo listando o tempo total de execução e o número total de chamadas de todas as funções.
  2. Uma listagem das funções ordenadas pelo tempo gasto por cada uma.
- Exige que o programa tenha sido compilado com a flag ***-pg*** para funcionar:

> Quando o parâmetro ***-pg*** é passado para o compilador, ele insere um contador após cada sub-rotina, permitindo gerar as estatísticas.

Preparando o programa

Forma correta de compilar e usar

```
$ gcc sample1.c -o sample -pg
$ ./sample
$ gprof -b sample > dump.txt
```

- Note que é necessário executar o programa.
- Quando o programa é executado, ele gera o arquivo ***gmon.out***, que contém as informações usadas pelo **gprof**.

Parâmetro *-pg*

Obtendo o assembly

```
$ gcc -S -pg sample1.c -o s1profiled
$ gcc -S sample1.c -o s1NOTprofiled
```

- ***-pg*** insere a chamada à função ***mcount***.
- Esta função que gera as informações salvas em ***gmon.out***. Fora isso, todas isntruções serão idênticas (o que é esperado).

Diff

```
$ diff s1profiled.s s1NOTprofiled.s -y --suppress-common-lines
    call mcount <
    call mcount <
    call mcount <
```

Verificando mcount

s1profiled

```
1 multiplica1:
2     pushl   %ebp
3     movl    %esp, %ebp
4     subl    $24, %esp
5     call    mcount          # UTILIZE ESTA CHAMADA EM SEUS CÓDIGOS ASM
6     movl    $0, -8(%ebp)
7     movl    $0, -4(%ebp)
8     jmp     .L4
```

s1NOTprofiled

```
1 multiplica1:
2     pushl   %ebp
3     movl    %esp, %ebp
4     subl    $24, %esp
5     movl    $0, -8(%ebp)
6     movl    $0, -4(%ebp)
7     jmp     .L4
```

Mensurando o programa

sample1.c

```
1  int soma(const int i, const int k) {
2      return i + k;
3  }
4
5  int multiplica1(const int i, const int j) {
6      int k, mul;
7      mul = 0;
8      for (k = 0; k < i; k++)
9          mul += soma(mul, j);
10     return mul;
11 }
12
13 int main(void) {
14     int i;
15     for (i = 0; i < 10000; i++)
16         soma(i, i);
17    for (i = 0; i < 10000; i++)
18         multiplica1(i, i);
19     return 0;
20 }
```

Mensurando o programa: Perfil Resumido

dump.txt

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 81.25      0.52     0.52    10000    52.00    64.00  multiplica1
 18.75      0.64     0.12 50005000     0.00     0.00  soma
```

Mensurando o programa: Call graph

dump.txt

```
granularity: each sample hit covers 4 byte(s) for 1.56% of 0.64 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.64                 main [1]
                0.52    0.12   10000/10000       multiplica1 [2]
                0.00    0.00   10000/50005000     soma [3]
-----------------------------------------------
                0.52    0.12   10000/10000       main [1]
[2]    100.0    0.52    0.12   10000         multiplica1 [2]
                0.12    0.00 49995000/50005000     soma [3]
-----------------------------------------------
                0.00    0.00   10000/50005000     main [1]
                0.12    0.00 49995000/50005000     multiplica1 [2]
[3]     18.8    0.12    0.00 50005000          soma [3]
-----------------------------------------------
```

Mensurando o programa: Indices

dump.txt

Index by function name
[2] multiplica1 [3] soma