COMPTE-RENDU

TP numéro 8 [LO22]

07 JUIN 2016

Marion Chan-Renous-Legoubin & Laviolette Etienne

Chan-Renous-Legoubin Marion
Laviolette Etienne

Le 11 Avril 2016

Rappel des objectifs du TP :

L'objectif principal de ce TP numéro 8 de LO22 est de nous introduire le concept de "profiling" d'application. Ceci permet de déterminer dans quelles portions du code le programme passe la plupart de son temps et à partir de quelles fonctions celles-ci sont invoquées. Ces informations permettent de révéler des problèmes d'optimisation, afin de pouvoir les améliorer. Nous allons donc effectuer le profiling d'une application JAVA, d'une application C, et d'un programme en c avec l'outil Valgrind.

I. Profiling d'application JAVA

Programme :

```
import java.util.*;
import java.io.*;
public class tp81{
    public static void main(String[] args) {
        for(int i=0;i<10;i++)
        {
        System.out.println(i);
        }
    }
}
```

Compilation et commandes java:

```
bash-4.2$ javac programme.java
programme.java:3: error: class tp81 is public, should be declared in a file name
d tp81.java
public class tp81{
       ^
1 error
bash-4.2$ java -prof programme
Erreur : impossible de trouver ou charger la classe principale programme
Dumping CPU usage in old prof format ... done.
bash-4.2$ java -Xrunhprof programme
Erreur : impossible de trouver ou charger la classe principale programme
Dumping Java heap ... allocation sites ... done.
```

Analyse des résultats : (fichiers java.prof et java.hprof.txt en annexe du rapport)

Fichier java.prof:

La première ligne de ce fichier nous indique : count callee caller time.

- Count correspond au nombre de fois où la fonction est appelée
- Callee renvoie à la fonction appelée
- Caller renvoie à la fonction appelante
- Time correspond au temps de l'exécution du programme java

Fichier java.hprof.txt:

Ce fichier nous apporte des informations sur l'execution du programme java.

Ces fichier servent ainsi à analyser l'optimisation du programme, détecter des appels infinis à des fonctions etc. En fonction de ces paramètres il est possible d'améliorer le programme.

II. Profiling d'application C

Programme initial :

```
/* * main.c */

#include <stdio.h>

char *AnotherString = "What are you gonna do, bleed on me?";

/* * WriteMyString.c */
extern char *AnotherString;
void WriteMyString(ThisString)
char *ThisString;
{
    int i=0;
    while(i<50)
    {
        printf("%s\n", ThisString);
        printf("Global Variable = %s\n", AnotherString);
        i++;
    }
}
main()
{
    int i;
    printf("Running...\n");
    for (i=0;i<5;i++)
    WriteMyString(AnotherString);
    printf("Finished.\n");
}
```

Compilation, exécution de l'exécutable et traitement gprof:

```
bash-4.2$ (./a.out)
Running...
What are you gonna do, bleed on me?
Global Variable - What are you gonna do, bleed on me?
What are you gonna do, bleed on me?
Global Variable - What are you gonna do, bleed on me?
What are you gonna do, bleed on me?
Global Variable - What are you gonna do, bleed on me?
What are you gonna do, bleed on me?
Global Variable - What are you gonna do, bleed on me?
What are you gonna do, bleed on me?
Global Variable - What are you gonna do, bleed on me?
What are you gonna do, bleed on me?
Global Variable - What are you gonna do, bleed on me?
What are you gonna do, bleed on me?
Global Variable - What are you gonna do, bleed on me?
What are you gonna do, bleed on me?
Global Variable - What are you gonna do, bleed on me?
What are you gonna do, bleed on me?
Global Variable - What are you gonna do, bleed on me?
What are you gonna do, bleed on me?
```

```
bash-4.2$ gprof -display
[main] debug-level=0x1
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        5     0.00     0.00  WriteMyString

 %         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
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.


                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       5/5           main [7]
[1]      0.0    0.00    0.00       5         WriteMyString [1]
-----------------------------------------------
```

Nous ajoutons une nouvelle boucle au programme.

<u>Nouveau programme:</u>

```
/* * main.c */

#include <stdio.h>

char *AnotherString = "What are you gonna do, bleed on me?";

/* * WriteMyString.c */
extern char *AnotherString;
void WriteMyString(ThisString)
char *ThisString;
{
    int i=0;
    while(i<2)
    {
        printf("%s\n", ThisString);
        printf("Global Variable = %s\n", AnotherString);
        i++;
    }
}
main()
{
    int j;
    printf("Running...\n");
    for (j=0;j<15;j++)
    WriteMyString("Bonjour Tiennou");

    printf("Finished.\n");
}
```

Compilation, exécution de l'exécutable et traitement gprof:

```
bash-4.2$ ./a.out
Running...
Bonjour Tiennou
Global Variable - What are you gonna do, bleed on me?
Bonjour Tiennou
Global Variable - What are you gonna do, bleed on me?
Bonjour Tiennou
Global Variable - What are you gonna do, bleed on me?
Bonjour Tiennou
Global Variable - What are you gonna do, bleed on me?
Bonjour Tiennou
Global Variable - What are you gonna do, bleed on me?
Bonjour Tiennou
Global Variable - What are you gonna do, bleed on me?
Bonjour Tiennou
Global Variable - What are you gonna do, bleed on me?
Bonjour Tiennou
Global Variable - What are you gonna do, bleed on me?
Bonjour Tiennou
Global Variable - What are you gonna do, bleed on me?
Bonjour Tiennou
Global Variable - What are you gonna do, bleed on me?
Bonjour Tiennou
```

```
bash-4.2$ gprof -display
[main] debug-level=0x1
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00       15     0.00     0.00  WriteMyString

 %         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
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.


                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00      15/15          main [7]
[1]      0.0    0.00    0.00      15         WriteMyString [1]
-----------------------------------------------

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.
```

Le traitement gprof du programme nous sert à savoir combien de fois une boucle a été appelée dans un programme. Ceci peut être utile afin d'analyser un programme et de l'optimiser, de comparer le

nombre d'appels à chaque fonction avec le nombre d'appels effectivement utiles, afin d'en diminuer le nombre et d'ainsi augmenter l'efficacité du programme.

III. Valgrind

Programme:

```c
#include <stdlib.h>
void f(void)
{
    int* x = malloc(10 * sizeof(int));
    x[10] = 0;
}
int main(void)
{
    f();
    return 0;
}
```

Analyse du programme avec l'outil Valgrind

Valgrind permet d'analyser la mémoire utilisée par le programme. Ainsi, cette analyse permet de détecter des anomalies au niveau de l'exécution (par exemple une exécution trop lente), des erreurs d'allocation et de désallocation de mémoire (fuites de mémoire possibles).

Point de vue sur ces types d'outils

Les trois outils utilisés dans ce TP ont chacun des intérêts propres vis-à-vis de l'analyse de programmes informatique. Ceux-ci permettent d'optimiser les programmes afin de les rendre les plus performants possible. Nous avons trouvé que ces programmes peuvent se révéler très utiles, notamment dans la détection de bugs qui peuvent être transparents au moment de la compilation. Pour de très gros programmes, ces outils nous paraissent indispensables afin que ceux-ci fonctionne correctement et efficacement.