

Practical 3

AIM: Demonstration of gprof command using Linux.

THEORY:

The objective of profiling is to analyze your program code and see which part of the code is taking a large amount of time for execution such that the part of the code can be rewritten. This will enable the program to achieve desired execution speed. In addition, profiling can prove to be very handy in spotting codes that are potentially error-prone, and then they can be sorted out via refactoring.

Using the gprof is quite simple. First, you need to enable profiling when you compile the code. Now, when you execute the program, profile data is generated. Finally, you can run the gprof tool on the profiling data generated during execution. This will produce an analysis file. This file contains several tables, including a flat profile and a call graph.

Enable profiling while compiling

To enable profiling while compiling, simply add the `-pg` option in the compilation step. The `pg` flag generates extra code for profiling that is used by the `gprof` command. The following command illustrates how we can compile profiling information:

```
$ gcc -Wall -pg program.c program_new.c -o program
```

Execute the code to generate profile information

The binary file generated above can be executed to generate profile information. The following line executes the code:

```
$/program
```

This will also generate a file `gmon.out` in the current working directory. To see what files are generated via execution of the above command, simply type `ls` as follows:

```
$ls
```

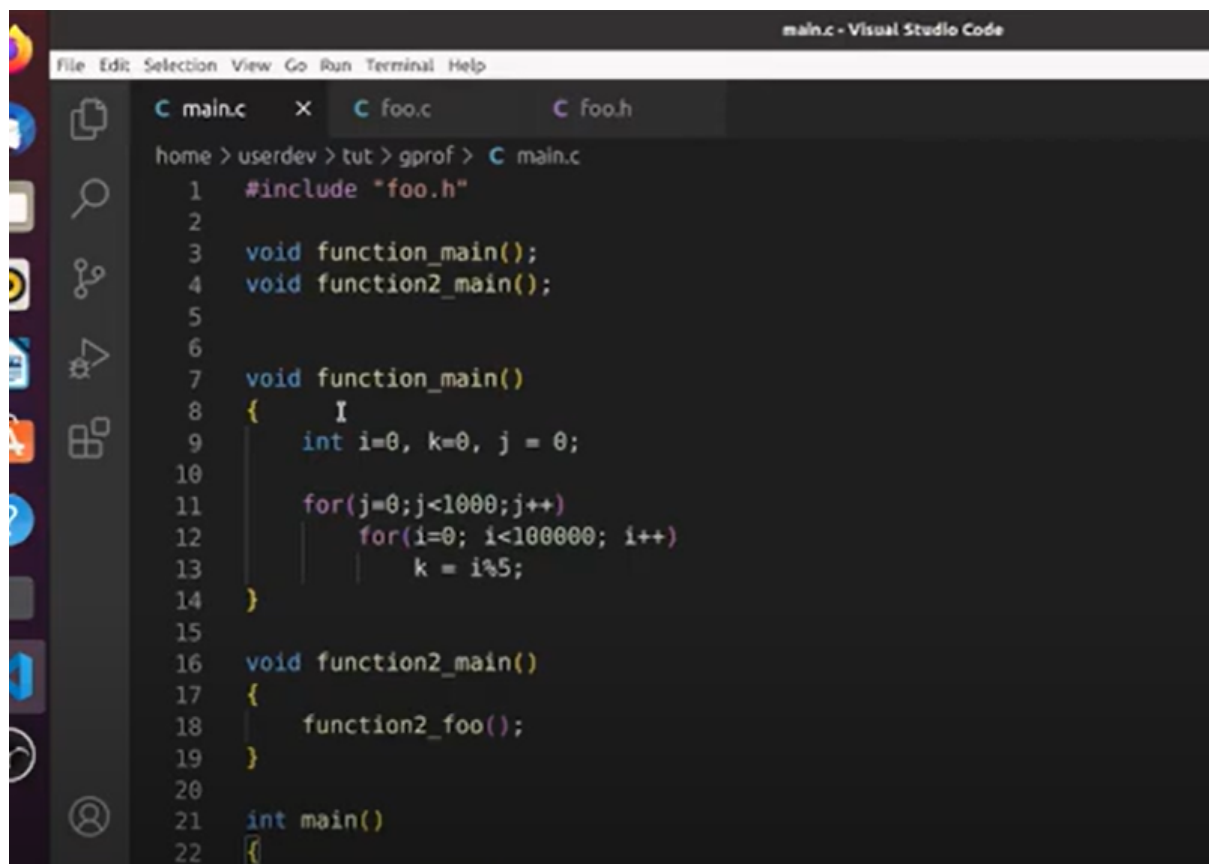
Run the gprof tool

Now, we will run the `gprof` tool providing as an argument the output file and `gmon.out` file. This will produce the profiling information:

```
$ gprof program gmon.out > analysis.txt
```

PROGRAM CODE

`main.c` file

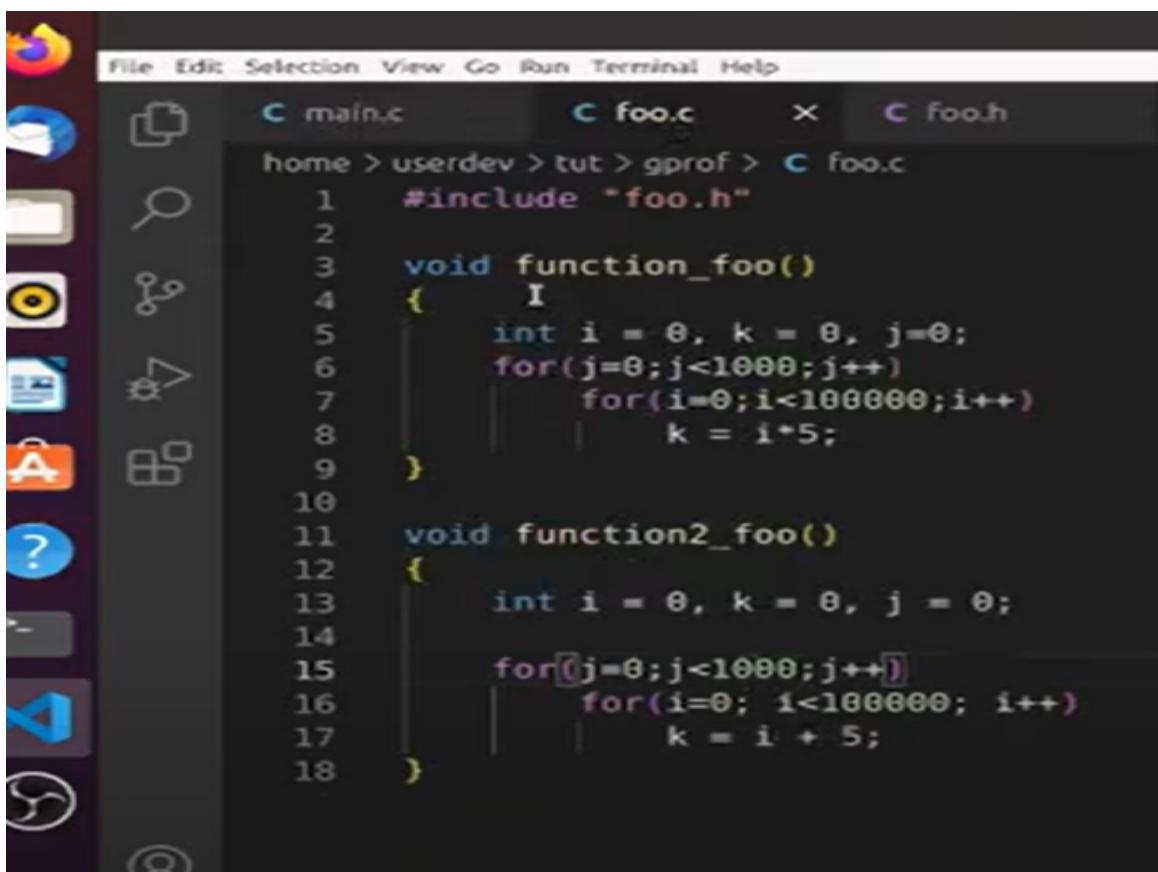


```
main.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help

C main.c x C foo.c C foo.h
home > userdev > tut > gprof > C main.c
1  #include "foo.h"
2
3  void function_main();
4  void function2_main();
5
6
7  void function_main()
8  {
9      I
10     int i=0, k=0, j = 0;
11
12     for(j=0;j<1000;j++)
13         for(i=0; i<100000; i++)
14             k = i%5;
15
16 void function2_main()
17 {
18     function2_foo();
19 }
20
21 int main()
22 {
```

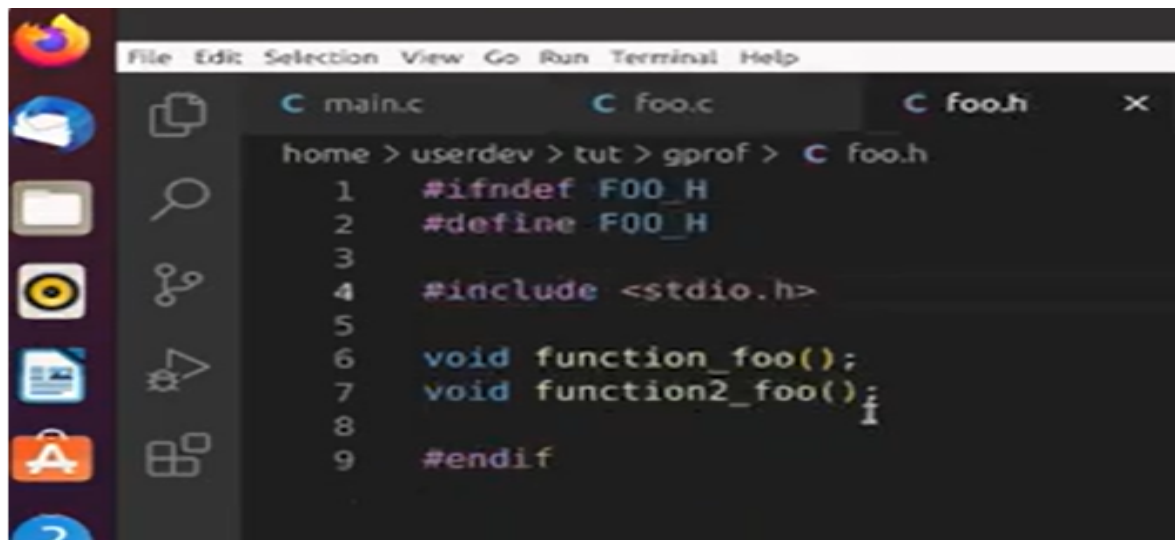
```
21 int main()
22 {
23     function_main();
24     function2_main();
25     function_foo();
26
27     return 0;
28 }
```

foo.c file:



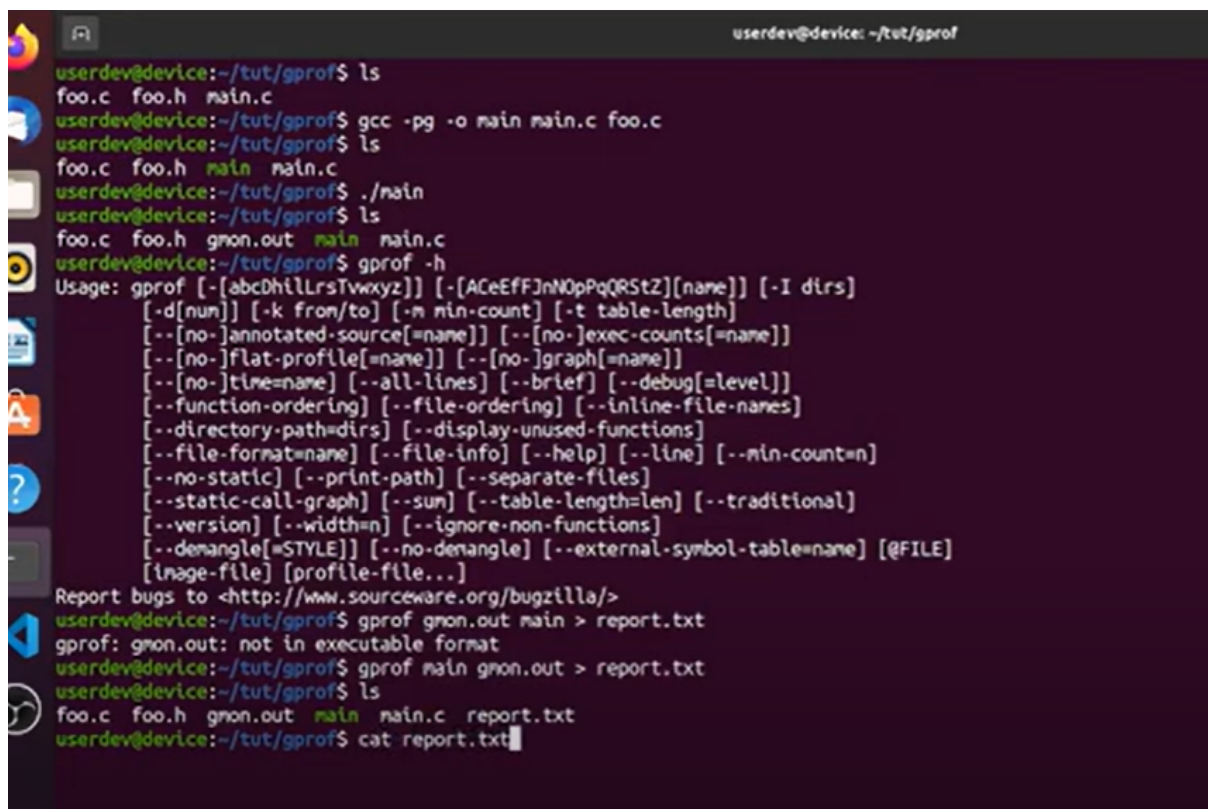
```
File Edit Selection View Go Run Terminal Help
C main.c C foo.c x C foo.h
home > userdev > tut > gprof > C foo.c
1 #include "foo.h"
2
3 void function_foo()
4 {
5     int i = 0, k = 0, j=0;
6     for(j=0;j<1000;j++)
7         for(i=0;i<100000;i++)
8             k = i*5;
9 }
10
11 void function2_foo()
12 {
13     int i = 0, k = 0, j = 0;
14
15     for(j=0;j<1000;j++)
16         for(i=0; i<100000; i++)
17             k = i + 5;
18 }
```

Foo.h file



```
File Edit Selection View Go Run Terminal Help
C main.c C foo.c C foo.h x
home > userdev > tut > gprof > C foo.h
1  #ifndef FOO_H
2  #define FOO_H
3
4  #include <stdio.h>
5
6  void function_foo();
7  void function2_foo();
8
9  #endif
```

OUTPUT:



```
userdev@device: ~/tut/gprof
userdev@device:~/tut/gprof$ ls
foo.c foo.h main.c
userdev@device:~/tut/gprof$ gcc -pg -o main main.c foo.c
userdev@device:~/tut/gprof$ ls
foo.c foo.h main main.c
userdev@device:~/tut/gprof$ ./main
userdev@device:~/tut/gprof$ ls
foo.c foo.h gmon.out main main.c
userdev@device:~/tut/gprof$ gprof -h
Usage: gprof [-[aBcDhLLrsTWxyz]] [-[ACeEfF]nNOpPq[QRStZ][name]] [-I dirs]
        [-d[num]] [-k from/to] [-m min-count] [-t table-length]
        [--[no-]annotated-source[=name]] [--[no-]exec-counts[=name]]
        [--[no-]flat-profile[=name]] [--[no-]graph[=name]]
        [--[no-]time[=name]] [--all-lines] [--brief] [--debug[=level]]
        [--function-ordering] [--file-ordering] [--inline-file-names]
        [--directory-path=dirs] [--display-unused-functions]
        [--file-format=name] [--file-info] [--help] [--line] [--min-count=n]
        [--no-static] [--print-path] [--separate-files]
        [--static-call-graph] [--sun] [--table-length=len] [--traditional]
        [--version] [--width=n] [--ignore-non-functions]
        [--demangle[=STYLE]] [--no-demangle] [--external-symbol-table=name] [%FILE]
        [image-file] [profile-file...]
Report bugs to <http://www.sourceware.org/bugzilla/>
userdev@device:~/tut/gprof$ gprof gmon.out main > report.txt
gprof: gmon.out: not in executable format
userdev@device:~/tut/gprof$ gprof main gmon.out > report.txt
userdev@device:~/tut/gprof$ ls
foo.c foo.h gmon.out main main.c report.txt
userdev@device:~/tut/gprof$ cat report.txt
```

