NAME

gprof – display call-graph profile data

SYNOPSIS

gprof [**–abcCDlsz**] [**–e** *function-name*] [**–E** *function-name*] [**–f** *function-name*] [**–F** *function-name*] [*image-file* [*profile-file* . . .]] [**–n** *number of functions*]

DESCRIPTION

gprof produces an execution profile of a program. The effect of called routines is incorporated in the profile of each caller. The profile data is taken from the call graph profile file which is created by programs compiled with the **–xpg** option of **cc**(1B), **–pg** for other compilers, or by setting the **LD_PROFILE** environment variable for shared objects (see **ld**(1)). These compiler options also link in versions of the library routines which are compiled for profiling. The symbol table in the executable image file *image-file* (**a.out** by default) is read and correlated with the call graph profile file *profile-file* (**gmon.out** by default).

If more than one profile file is specified, the **gprof** output shows the sum of the profile information in the given profile files.

First, execution times for each routine are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle. The first listing shows the functions sorted according to the time they represent, including the time of their call graph descendants. Below each function entry is shown its (direct) call-graph children, and how their times are propagated to this function. A similar display above the function shows how this function's time and the time of its descendants is propagated to its (direct) call-graph parents.

Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle and their contributions to the time and call counts of the cycle.

Next, a flat profile is given, similar to that provided by **prof**(1). This listing gives the total execution times and call counts for each of the functions in the program, sorted by decreasing time. Finally, an index is given, showing the correspondence between function names and call-graph profile index numbers.

A single function may be split into subfunctions for profiling by means of the **MARK** macro (see **prof**(5)).

Beware of quantization errors. The granularity of the sampling is shown, but remains statistical at best. It is assumed that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call-graph arcs to parents of that function is directly proportional to the number of times that arc is traversed.

The profiled program must call **exit**(2) or return normally for the profiling information to be saved in the **gmon.out** file.

OPTIONS

–a Suppress printing statically declared functions. If this option is given, all relevant information about the static function (for instance, time samples, calls to other functions, calls from other functions) belongs to the function loaded just before the static function in the **a.out** file.

–b Brief. Suppress descriptions of each field in the profile.

–C Demangle C++ symbol names before printing them out.

–c Discover the static call-graph of the program by a heuristic which examines the text space of the object file. Static-only parents or children are indicated with call counts of 0.

–D Produce a profile file **gmon.sum** that represents the difference of the profile information in all specified profile files. This summary profile file may be given to subsequent executions of **gprof** (also with –**D**) to summarize profile data across several runs of an **a.out** file. (See also the –**s** option.)

As an example, suppose function A calls function B **n** times in profile file **gmon.sum**, and **m** times in profile file **gmon.out**. With –**D**, a new **gmon.sum** file will be created showing the number of calls from A to B as **n-m**.

–**E** *function-name*
Suppress printing the graph profile entry for routine *function-name* (and its descendants) as –**e**, below, and also exclude the time spent in *function-name* (and its descendants) from the total and percentage time computations. More than one –**E** option may be given. For example:

'–**E** *mcount* –**E** *mcleanup*'

is the default.

–**e** *function-name*
Suppress printing the graph profile entry for routine *function-name* and all its descendants (unless they have other ancestors that are not suppressed). More than one –**e** option may be given. Only one *function-name* may be given with each –**e** option.

–**F** *function-name*
Print the graph profile entry only for routine *function-name* and its descendants (as –**f,** below) and also use only the times of the printed routines in total time and percentage computations. More than one –**F** option may be given. Only one *function-name* may be given with each –**F** option. The –**F** option overrides the –**E** option.

–**f** *function-name*
Print the graph profile entry only for routine *function-name* and its descendants. More than one –**f** option may be given. Only one *function-name* may be given with each –**f** option.

–l Suppress the reporting of graph profile entries for all local symbols. This option would be the equivalant of placing all of the local symbols for the specified executable image on the –**E** exclusion list.

–n Limits the size of flat and graph profile listings to the top **n** offending functions.

–s Produce a profile file **gmon.sum** which represents the sum of the profile information in all of the specified profile files. This summary profile file may be given to subsequent executions of **gprof** (also with –**s**) to accumulate profile data across several runs of an **a.out** file. (See also the −**D** option.)

−z Display routines which have zero usage (as indicated by call counts and accumulated time). This is useful in conjunction with the −**c** option for discovering which routines were never called.

ENVIRONMENT

PROFDIR If this environment variable contains a value, place profiling output within that directory, in a file named *pid***.***programname*. *pid* is the process ID, and *programname* is the name of the program being profiled, as determined by removing any path prefix from the **argv[0]** with which the program was called. If the variable contains a null value, no profiling output is produced. Otherwise, profiling output is placed in the file **gmon.out**.

FILES

a.out	executable file containing namelist
gmon.out	dynamic call-graph and profile
gmon.sum	summarized dynamic call-graph and profile
$PROFDIR/*pid***.***programname*	

NOTES

If the executable image has been striped and it has no symbol table (.symtab) then gprof will read the dynamic symbol table (.dyntab) if present. If the dynamic symbol table is used then only the information for the global symbols will be available, the behavior will be identical to the −**a** option.

SEE ALSO

ld(1), **cc**(1B), **prof**(1), **exit**(2), **profil**(2), **monitor**(3C), **prof**(5)

Graham, S.L., Kessler, P.B., McKusick, M.K., '*gprof: A Call Graph Execution Profiler*', *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 17, No. 6, pp. 120-126, June 1982.

Linker and Libraries Guide

BUGS

Parents which are not themselves profiled will have the time of their profiled children propagated to them, but they will appear to be spontaneously invoked in the call-graph listing, and will not have their time propagated further. Similarly, signal catchers, even though profiled, will appear to be spontaneous (although for more obscure reasons). Any profiled children of signal catchers should have their times propagated properly, unless the signal catcher was invoked during the execution of the profiling routine, in which case all is lost.