NAME

gprof – display call graph profile data

SYNOPSIS

gprof [*options*] [a.out [gmon.out ...]]

DESCRIPTION

gprof produces an execution profile of a C, Pascal, or Fortran77 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 (*gmon.out* by default), which is created by programs compiled with the **–pg** option of *cc*, *pc*, and *f77*. The symbol table in the named object file (*a.out* by default) is read and correlated with the call graph profile file. If more than one profile file is specified, the *gprof* output shows the sum of the profile information in the given profile files.

First, a flat profile is given. This listing gives the total execution times and call counts for each of the functions in the program, sorted by decreasing time.

Next, these times 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. A second listing shows the functions sorted according to the time they represent including the time of their call graph descendents. 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 descendents is propagated to its (direct) call graph parents.

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

UNIVERSAL FILE SUPPORT

gprof accepts a ‘‘universal’’ file for the *a.out* file, using the host architecture from the file. (It is an error if the ‘‘universal’’ file does not contain the host architecture.)

OPTIONS

The following options are available:

–a suppresses the displaying of statically declared functions. If this option is given, all relevant information about the static function (such as 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 suppresses the displaying of a description of each field in the profile.

–c the static call graph of the program is discovered by a heuristic which examines the text space of the object file. Static-only parents or children are indicated with call counts of 0. (The **–c** option is currently not supported.)

–e *name*
suppresses the displaying of the graph profile entry for routine *name* and all its descendants (unless they have other ancestors that aren't suppressed). More than one **–e** option may be given. Only one *name* may be given with each **–e** option.

–E *name*
suppresses the displaying of the graph profile entry for routine *name* (and its descendants) as **–e**, above, and also excludes the time spent in *name* (and its descendants) from the total and percentage time computations. (For example, **–E** *mcount* and all of the other *monitor*(3) routines are excluded by default.)

–f *name*
displays the graph profile entry of only the specified routine *name* and its descendants. More than one **–f** option may be given. Only one *name* may be given with each **–f** option.

–F *name*
displays the graph profile entry of only the routine *name* and its descendants (as **–f,** above) and also uses only the times of the displayed routines in total time and percentage computations. More than one **–F** option may be given. Only one *name* may be given with each **–F** option. The **–F**

option overrides the **−E** option.

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

−S produces four order files suitable as input to *ld*(1): *gmon.order* is an ordering based on a closest is best algorithm, *callf.order* is based on call frequency, *callo.order* is based on call order and *time.order* is based on time. The order files contain only those functions which were called or sampled (including spontaneous functions). For library functions to appear correctly in the order file, a *whatsloaded* file produced by *ld*(1) should exist in the working directory. Filenames in the order file will be missing for: files compiled without the **−g** option, assembly files, and stripped executables. This option does not work with executables that have already been scattered. The *gmon.order* file can take a long time to produce and can be suppressed with the **−x** option.

−z displays 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.

FILES

a.out	the namelist and text space.
gmon.out	dynamic call graph and profile.
gmon.sum	summarized dynamic call graph and profile.
gmon.order	ordering based on closest is best algorithm.
callf.order	ordering based on call frequency.
callo.order	ordering based on call order.
time.order	ordering based on time.

SEE ALSO

monitor(3), profil(2), cc(1)
dyld(1) and the DYLD_IMAGE_SUFFIX environment variable
"gprof: A Call Graph Execution Profiler", by Graham, S.L., Kessler, P.B., McKusick, M.K.; *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 17, No. 6, pp. 120-126, June 1982.

BUGS

Beware of quantization errors. The granularity of the sampling is shown, but remains statistical at best. We assume 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.

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.

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