MPW PQR4 Proff and PrintProff
Release Notes

Introduction

Proff and PrintProff are a pair of components which provide profiling and performance monitoring for programs compiled from MPW C and MPW Object Pascal. *Profiling* is the dynamic recording for every procedure/function call during program execution, of the identity of the called procedure/function and the point from which it was called, e.g., statement `n` in procedure `foo`. *Performance monitoring* in the context of profiling is the recording of the time spent in each such procedure/function. (For the remainder of this Note, the term "procedure" will be used to mean "procedure or function.") An *arc* is defined to be an execution of a procedure from a specific call site. For each arc is recorded:

1.The number of times it was executed.

2.The cumulative *flat* time for the arc. The flat time is the amount of time spent executing the procedure. It does not include time spent in callee procedures, provided these procedures were also monitored. Nor does it include segment loading or profiling overhead for the current procedure but it does include some performance monitoring and procedure call overhead for the procedure (if any) called directly by the current one. Note: If a callee was not built with the profiling options or directives, the profiling code will think that the callee was merely part of its caller.

3. The cumulative *hierarchical* time for the arc. The hierarchical time is the amount of time spent in this procedure, plus the time spent in any procedures called by this one (directly or indirectly) before it returns to its caller. It does not include segment loading or profiling overhead for the current procedure. However, the hierarchical time does include this overhead for called procedures.

4. For calls across segments, the caller and the callee segment name and number.

Proff is a library (literally, Proff.o, contained in {MPW}Libraries:Libraries:) which contains the code that gathers the profiling and performance data. In order to achieve the data collection, it is necessary to compile the subject programs with the appropriate options and directives, the details differing slightly between C and Object Pascal, and to include Proff.o in the link.

PrintProff is an MPW tool that processes the data file created by Proff and produces a human readable output.

Program Building

For both C and Object Pascal, specify `-sym full` on the command line. For C, specify `-trace on`, or bracket the code to be monitored with `#pragma trace on` and `#pragma trace off`. For Object Pascal, bracket the code to be monitored with `{$D++}` and `{$D--}`. Include both StdClib.o and Proff.o in the link—StdClib.o is required even if all of the source is in Object Pascal. Specify `-sym full` on the Link command line. In rare cases, the C compiler may optimize away the use of the `A6` links which are required by `proff` to find the right caller. To ensure that this does not occur, use `-opt nodelink`.

Execution Monitoring

Execution of a program that was built for monitoring will cause the creation of an output file containing the performance data. This file will be in the same directory as the target program and will have the application name with .Proff appended as a suffix.

△ **Important:** MPW tools create difficulties for application oriented debuggers and performance monitors. The application is the *MPW Shell*, so the name of the output file will be 'MPW Shell.Proff'. The user should change this name to <toolname>.Proff. △

Although the output file is intended to be processed by PrintProff, its format may be of interest. It consists of one-line records containing the following information: caller segment number and byte offset, callee segment number and byte offset, number of times the arc executed, cumulative hierarchical time, and cumulative flat time. The last line of the output file is simply a count of the number of arcs in the file. All numbers are given in hex.

An example follows:

```
     1      64       1       c       1  2fdbc0   24ef0
1     134       1      a6       1  31db97   9076a
1     1de       2       e       1   1bca7     e75
1     270       3       c       d 59076a4   8ecad
1     29c       7       c       3   2abe7    464c
3      50       5      10       4  b5a754    8eca
3      56       5     25a       3 37fa85c   076a4
3      5c       1     298       3   d2568    65b1
3      4a       4     14a       3 1404055   5b1b6
4      a2       4      10      40 18ecad9    9076
4     15a       7       c       3   226bb    1404
4     182       8      50       7   464ca    5a75
4     1ec       4      10       7  65b1b6     85c
4     206       7      4c       2    90f9     5a7
e
```

Using PrintProff

PrintProff is an MPW tool which analyzes the .proff data file created by running a program built for profiling. PrintProff uses as input the target program's .proff and .sym files. By default, both of these files have the same name as the target program with the appropriate extension added. Also by default, PrintProff looks for them in the Shell's current directory. The output from PrintProff goes to standard output unless redirected to an output file. To run PrintProff enter:

```
PrintProff TargetProgram
```

If the .proff and .sym files are in the current directory, `TargetProgram` is just the target's terminal name. If they are in some other directory (they must both be in the same one), `TargetProgram` must be a complete path including the terminal name.

An example of the output from PrintProff follows:

```
1. main
        Total:  22,103,748  H.µsecs.  185,050  F.µsecs. /1 call
        (100.000 %H/ 0.837 %F)
        Callers:
         0.  %__MAIN
                22,103,748  H.µsecs.  185,050  F.µsecs. /1 call

2.  SkelMain
        Total:  21,071,898  H.µsecs.  5,754,715  F.µsecs. /1 call
                (95.332 %H/26.035 %F)
        Callers:
         1.  main.(51)
                21,071,898  H.µsecs  5,754,715  F.µsecs.  /1 call
                        1/Main -> 3/TransSkel

3. DoEvent
        Total:  5,574,969  H.µsecs  3,168,383  F.µsecs. /63 calls
                (25.222 %H/14.334 %F)
        Callers:
         2.  SkelMain.(15)
                5,574,969  H.µsecs.  3,168,383  F.µsecs.  /63 calls

4. LogEvent
        Total:  5,489,880  H.µsecs.  101,050  F.µsecs. /63 calls
                (24.837 %H/ 0.457 %F)
        Callers:
         2.  SkelMain.(14)
                5,489,880  H.µsecs.  101,050  F.µsecs.  /63 calls
                        3/TransSkel -> 1/Main

5. DisplayText
        Total:  4,546,138  H.µsecs.  4,128,112  F.µsecs. /225 calls
                (20.567 %H/18.676 %F)
        Callers:
         7.  DisplayString.(1)
                2,601,034  H.µsecs.  2,371,566  F.µsecs.  /120 calls
         8.  DisplayChar.(1)
                1,837,651  H.µsecs.  1,651,699  F.µsecs.  /104 calls
         1.  main.(15)
                107,451  H.µsecs.        104,846  F.µsecs.  /1 call
                        1/Main -> 3/TransSkel
```

6. Background

Total: 3,823,245 H.µsecs. 785,805 F.µsecs. /226 calls

(17.297 %H/ 3.555 %F)

Callers:

2. SkelMain.(7)

3,823,245 H.µsecs. 785,805 F.µsecs. /226 calls

3/TransSkel -> 1/Main

Known Bugs and Limitations

- Because of conflicting uses of VIA Timer1, Proff.o cannot be applied to applications that use the Sound Manager or the Time Manager.
- The Object Pascal compiler fails to insert the required profiling calls in the main program block, despite a request that it so do. The workaround is to enclose the actual main block by a dummy block.
- Proff.o assumes that procedures being profiled , and their call sites, are located in resources of type 'CODE' in the resource map of the target application with which Proff.o is linked. Any resources added to the map after Proff.o first executes will not be monitored.
- Currently, Proff.o needs two routines from StdClib.o (`printf` and `sprintf`). If the target program is Pascal, and isn't already using StdClib.o, this must be linked in last.
- The times reported for a recursive routine, as reached from each of its callers other than itself, are correct. However, the total hierarchical time reported for a directly or indirectly recursive routine is too large because of the way in which direct or indirect calls of a routine from itself are summed and included in the total. An investigation of this behavior turned out to be sufficiently interesting that a discussion of the matter is presented as Appendix C of these Release Notes.
- Calls to `exit()` in C do not execute the function epilogues of any functions that have yet to return. This results in zero time being reported for execution of those functions.
- The procedure `ExitToShell` does not call any of the exit routines that had been installed by `atExit`. Such an exit routine is used by Proff.o to write its output file. Therefore, if the host program calls `ExitToShell`, the output file will not be generated.

- Proff.o allocates its memory from Temporary Memory. If Temporary Memory is not available or insufficient then Proff.o will allocate its memory from the heap of the target application. If this is also insufficient, proff.o will drop into the debugger and ask you to increase the application heap size. Proff.o allocates all its memory the first time that a monitored procedure executes and will not move memory while the target program executes.

- Proff.o does not yet monitor patch or interrupt handling code, or any code which executes when `A5` does not belong to the application that was built for monitoring.