To: MTB Distribution

From: David Spector

Date: 14 May 1979

Subject: New Profile Command

Motivation

The present profile command has been extended by the private, uninstalled commands -long_profile (which accumulates statistics on execution counts, elapsed times, and page faults), plot_profile (which plots profile data on any Multics graphics display device), create_cost_listing (which lists source programs with associated profile data) and a version of profile that works with hardcore programs; these tools are all useful and should be incorporated into the standard profile command. A means of creating standard-format profile data segments is also desirable, so that performance studies of various versions of subject programs run on various test cases can be handled systematically and can produce tangible data output in the form of data segments, rather than the present availability of profile data only within the lifetime of a single process.

New MPM Documentation

Name: profile

The profile command is a performance measuring tool that analyzes the time spent executing each source statement of a program, along with other parameters of interest, after the program is run.

The program to be analyzed must be compiled using the -profile (-pf) control argument of the cobol, fortran, and pl1 commands, or using the -long_profile (-lpf) control argument of the pl1 command. The long_profile compiler option is used to acquire exact elapsed time statistics and is more expensive to use than the -profile compiler option.

Usage

profile {program_names} {-control_args}

where:

1. program_names
are entrynames or reference names of programs to be analyzed. They need not be specified if the -input_file control argument is used (see below).

2. control_args
are selected from the following list. Control arguments apply to all programs specified, and may be given in any order.

-print, -pr
prints the following information for each statement in the specified program(s):

1. Line number.
2. Statement number (if greater than 1).
3. Count: the number of times the statement was executed.
4. Cost: an approximation to the accumulated execution time for the statement. Equal to the number of instructions executed plus ten times the number of external operators called.
5. Names of all external operators called by the statement.

For -long_profile (actual accumulated time) data, items 4 and 5 are changed to the following:

4. Time: actual execution time for the statement in virtual CPU microseconds.
5. Faults: page faults incurred in executing the statement.

-sort STR
used with -print to sort profile information into descending order of the specified field STR, which may be any one of the following:

count	time
cost	faults

-long, -lg
used with -print to include in the output information for statements that have never been executed.

-list path, -ls path
creates a profile listing of the source segment specified by path, which must include the language suffix. The profile listing file is given the list suffix, and is created in the working directory. The listing includes the information described above for the -print control argument, as well as a column of stars (asterisks) indicating the percentage of total cost/time according to the following scheme:

4 stars: 20% to 100%
3 stars: 10% to 20%
2 stars: 5% to 10%
1 star: 2.5% to 5%
no stars: 0% to 2.5%

-exclude STR, -ex STR
used with -list to exclude the column of information indicated by the field STR, which may be any one of the following:

stars	cost	operators, ops
line	time	
count	faults	

-line_length N, -ll N
used with -list to specify an output width of N characters. If not specified, N is assumed to be 132.

-plot STR
plots a line graph (on the user's graphics terminal) of the values of the specified field STR, which may be any one of the following:

count	time
cost	faults

-from N, -fm N
used with -plot to begin the plotting with the data for line number N. If -from is not specified, N is assumed to be 1.

-to N
used with -plot to end the plotting with the data for line number N. If -to is not specified, N is assumed to be the line number of the last executable statement.

-output_file path, -of path
causes the profile data for the specified program names to be stored in the profile data file specified by path. The file is created if it does not already exist. The pf suffix is added to path if it is not

already present. The profile data is stored in a format acceptable to the -input_file control argument (see below). The format of pf data files is described by the pl1 include file pf_format.incl.pl1. The stored data is determined by the program_names specified, as well as by the -comment control argument and whether the compilation was done using the -profile or -long_profile options.

-comment STR, -cm STR
used with the -output_file control argument to include STR with the stored profile data as a comment. If STR is to include blanks or other characters recognized as special by the command processor, it should be enclosed in quotes. STR may be up to 128 characters long.

-input_file path, -if path
causes the profile data to be retrieved from the profile data file specified by path. Use of this control argument causes the current (internal static) profile data, if any, to be ignored. The pf suffix is appended to path if it is not already present. If any program_names are specified, they select a subset of the stored data for analysis. If no program_names are specified, all data stored in the profile data file is used. This control argument may not be given if -output_file is specified.

-reset, -rs
resets (zeros) all current (internal static) profile data for the named program(s). The resetting is done as the final operation if -print, -list, -plot, or -output_file are also specified. This control argument may not be given if the -input_file or -hardcore control arguments are specified.

-hardcore, -hard
indicates that the specified programs are supervisor (hardcore) segments. The current (internal static) profile data for such programs is retrieved from the address space of the supervisor. Hardcore programs compiled with the -profile (or -long_profile) control argument must be installed by generating a Multics System Tape and rebooting Multics. See Multics System Programming Tools (AZ03) for a description of the generate_mst command. Note that the current (internal static) profile data for hardcore programs cannot be reset (zeroed).

-search path, -srh path
used with -hardcore to add the directory path to an internal search list of hardcore object directories. Up

to 8 directories may be specified. If no search list is specified, >ldd>hard>o is searched for copies of the specified program(s).

Notes

If none of the control arguments -print, -list, -plot, -output_file, or -reset are specified, -print is assumed.

When analyzing several runs of the same program(s) on various test cases, -reset should be specified. If -reset is not specified, the current (internal static) profile data is accumulated (added) for all runs.

There are two forms of profile data, current and stored. Current data is in a form suitable for direct incrementing by the program(s) being analyzed and is stored using the pl1 internal static storage class (or, in the case of hardcore programs, in a special hardcore data segment). Current profile data (except for hardcore programs) can be reset by the -reset control argument. Stored profile data is permanent data as stored by the -output_file control argument.

Profile listing and data files are automatically stored as multi-segment files (MSFs) if they are too large to fit into a single segment. This feature allows very large bound object segments to be analyzed and very large source segments to be listed.

Examples

quad; profile quad
prints the current profile data of the program quad. Note that quad must first be executed, in order to acquire current profile data.

profile quad -of quad
stores the current profile data in segment quad.pf.

profile -if quad
prints the stored profile data from quad.pf.

profile -if quad -list quad.fortran
creates profile listing quad.list from the source quad.fortran and the profile data quad.pf.

Profile Data File Format

```
/*        BEGIN INCLUDE FILE ... pf_format.incl.pl1 ... D. Spector May, 1979 */

/*        Format of profile data segments */

dcl       1 pf_header           aligned based (pf_ptr),      /* Start of pf segment */
            2 version           fixed bin,                   /* See pf format version below */
            2 date_time_stored
                                fixed bin (71),
            2 person_project    char (32),
            2 comment           char (128),
            2 first_program,                                 /* Msf offset in pf data to first
                                                                program data */
              3 component       fixed bin,
              3 offset          fixed bin (18),
            2 operator_array,                                /* Msf offset in pf data to
                                                                operator_array */
              3 component       fixed bin,
              3 offset          fixed bin (18);

/*        Data for one program or component */

dcl       1 program             aligned based (program_ptr), /* Profile data for a program */
            2 next_program,                                  /* Msf offset in pf data to next
                                                                program data */
              3 component       fixed bin,
              3 offset          fixed bin (18),
            2 name              char (32),                   /* Program name (does not include a
                                                                language suffix) */
            2 translator        char (8),                    /* Language name */
            2 flags,
              3 long_profile    bit (1) unal,
              3 mbz             bit (35) unal,
            2 n_values          fixed bin,
```

```
          2 value           (1 refer (program.n_values)),
            3 source,
              4 file        fixed bin (10) unsigned unal,
              4 line        fixed bin (16) unsigned unal,
              4 statement   fixed bin (5) unsigned unal,
              4 mbz         bit (5) unal,
            3 first_operator
                            fixed bin (19) unsigned unal, /* Subscript of first of list of
                                                             operators for this statement */
            3 n_operators   fixed bin (17) unsigned unal, /* Number of operators in list for
                                                             this statement */
                                                          /* If n_operators = 1,
                                                             first_operator contains */
                                                          /* the operator itself (to save
                                                             space) */
            3 count         fixed bin (35),               /* Execution count */
            3 cost_or_time  fixed bin (35),               /* Instructions or VCPU time
                                                             (long_profile) */
            3 page_faults   fixed bin (35);               /* (long_profile only) */

/*        Packed array of operators referenced by the program. Each operator consists
          of the offset into the operators specified by program.operators_name */

dcl       operator_array    (522240) fixed bin (18) unsigned unal based (operator_ptr);

dcl       pf_ptr            ptr;                        /* Pointer to base of pf segment
                                                           (component 0) */
dcl       pf_format_version_1 fixed bin int static options (constant) init (1);
dcl       program_ptr       ptr;                        /* Pointer calculated from
                                                           pf_header.first_program or
                                                           program.next_program */
dcl       operator_ptr      ptr;                        /* Pointer calculated from
                                                           pf_header.operator_array */

/*        END INCLUDE FILE ... pf_format.incl.pl1 */
```