A A two pages long profiler

The real-time profiler records entry and exit time intervals which are set by the programmer. These intervals can be nested, in fact all intervals are nested within the first interval which is recorded by the profiler itself. The profiler provides the following information:

code - the numeric code of the interval.

parent - the numeric code of the parent interval

description - the name/description of the interval

Total Time - Accumulated Time (without sibling intervals time) for the interval.

percentage - percentage of the elapsed time.

average time - average time per call.

calls - total number of entries to the interval (not including return from siblings).

The data is recoded using the START_PROFILE, END_PROFILE, ENTER(code), LEAVE(code) macros (windows version) and the recorded data is analyzed using a stack. Source listing follows.

A.1 Timing Macros (PC WINDOWS)

The following macros should be added to the program to be profiled. profiling starts at START_PROFILING and stoppes a STOP_PROFILING. ENTER(code) and LEAVE macros are used to bound sections timing. ENTER and LEAVE can be nested.

```
/* ==============================================
 * ============= Timing Macros (PC WIN) =========
 * ==============================================*/

#ifdef PROFILE

FILE *logfp, *recfp;
static _timeb tbuf;

#define START_PROFILE { \
    recfp = fopen("recording.dat","wb");\
        assert(recfp != NULL); \
    ENTER(0); \
}

#define STOP_PROFILE  { \
    LEAVE(0); \
    fclose(recfp); \
}

#define ENTER(x) { \
    _ftime(&tbuf); \
    fprintf(recfp,"E %d %d %d\n",(x),tbuf.time, tb uf.millitm); \
}

#define LEAVE(x) { \
    _ftime(&tbuf); \
    fprintf(recfp,"L %d %d %d\n",(x),tbuf.time, tb uf.millitm); \
}
```

```
#else

#define START_PROFILE
#define STOP_PROFILE
#define ENTER(x)
#define LEAVE(x)

#endif
```

A.2 Section codes and names

The following table defines the names of each section for the profiler.

```
/* ==============================================
 * ============= File: names.h ==================
 * ==============================================*/

#define CODES 5

char *name[] = {
/* 0 */ "Code 0",
/* 1 */ "Code 1",
/* 2 */ "Code 2",
/* 3 */ "Code 3",
/* 4 */ "Code 4",
/* 5 */ "Code 5",
};
```

A.3 profiler

The following is the off line profiler report program. It reads the recoding file from the standard input and produces a small report to the standart output.

```
/* ==============================================
 * =============File: Profiler.c ================
 * ==============================================*/

/* Simple Real-Time Profiler
 * ========================
 * Usage: profile < recording.dat
 *
 * written by moshe ben ezra - the hebrew university of jerusalem
 * www.cs.huji.ac.il/~moshe
 */

#include <stdio.h>
#include "names.h"

#define STACK_SIZE  100
#define MAX_CODES 50

main()
{
  static int stack[STACK_SIZE], sp;      /* Nested Calls stack and stack pointer */

  double
    ts, te,                              /* Start time, End time      */
    t1, t2,                           /* Interval time var s       */
    sum[MAX_CODES],                   /* sum time intervals        */
    dt;                               /* Temporary delta time var  */

  long
    sec,msec,                         /* Time from input file      */
    calls[MAX_CODES],                 /* Call counters             */
    parent[MAX_CODES];                /* Parent code               */

  int idx,i;                          /* Function index, loop i    */
  char code[2];                       /* Enter Exit code           */

  /*
   * Initialization
   */

  for(i=0; i<MAX_CODES; i++){
    sum[i] = calls[i] = parent[i] = 0;
  }

  for(i=0; i<STACK_SIZE; i++){
    stack[i] = 0;
  }

  ts = 0;
  sp = 1;                             /* 'Enter' always leaves the  */
  parent[0] = -1;                     /* the previsous level        */

  /*
   * process data file
   */

  while(scanf("%s%d%d%d\n",&code,&idx,&sec,&msec) == 4){
    t2 = (double) sec + (double) msec / 1000.0;         /* current time               */
    te = t2;                                            /* update end time            */
    switch (code[0]){

      /*===================*
       * Enter new section
       *===================*/
    case 'E':
      if (ts == 0) ts = t1 = t2;       /* At init prev interval = 0 */
      /*
       * leave prev level
       */
      dt = t2 - t1;                   /* Update Previous level interval   */
      sum[stack[sp-1]] += dt;         /* time (but not call counter)      */
      /*
       * enter new level
       */
      if (idx != 0)
    parent[idx] = stack[sp-1];        /* Save parent code               */
      stack[sp++] = idx;              /* Save section index             */
      t1 = t2;                        /* Update start time              */
      break;

      /*======================*
       * Leave current section
```

```
        *=====================*/
    case ’L’:
      if (idx != stack[sp-1]){          /* Coherence check                */
        fprintf(stderr,"Stack Mismatch !\n");
        exit(1);
      }

      dt = t2 - t1;                     /* Update time and calls counter   */
      sum[stack[sp-1]] += dt;
      calls[stack[sp-1]] ++;

      /*
       * Return to previous level
       */
      sp --;
      t1 = t2;                          /* Start new interval */
      break;
    default:
      fprintf(stderr,"Code Mismatch !\n");
      exit(1);
    }
  }

  dt = te - ts;                         /* Elapsed time */

  /*
   * Outpout simple report
   */
  printf("Duration: %g sec\n",dt);
  for(i=0; i<= CODES; i++)
    printf("Code: %2d  Parent: %2d (%-20s): %5.2f sec.  %6.2f%%      Avr: %5.2f msec  calls: %d\n",
       i,parent[i],name[i],sum[i],sum[i]/dt*100.0,sum[i]*1000.0/calls[i],calls[i]);
}
```