CSI605

Program Timing and Profiling

Simple Timers

- **Measure the total time that the system spent executing a program along with CPU usage statistics.**

- **time**
  - **A timer built into the UNIX C Shell**

- **/bin/time**
  - **Another timer that is not built into the UNIX C shell**

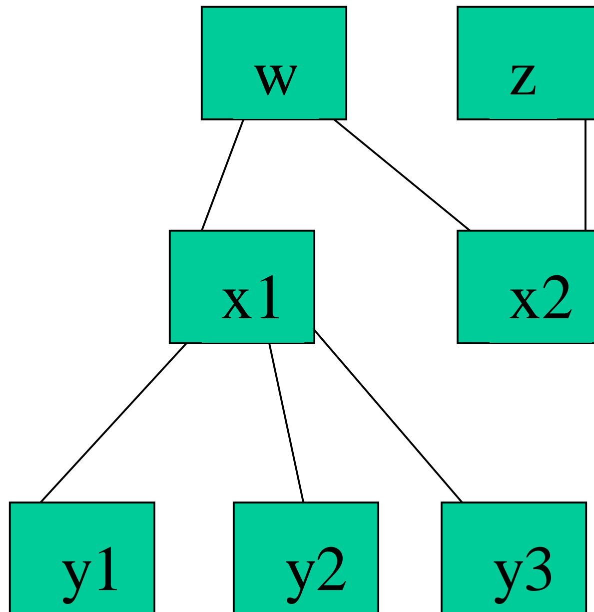
Report-Generating Profilers

- **These analyze program performance on a routine by routine basis**
- **Help identify bottle necks in code**
- **prof**
  - **One of the very early profilers**
- **gprof**
  - **A more current update GNU oriented profiler**
  - **Originally released by Berkeley**

Examples of Simple Timings

- **It is highly accurate**
- **Provides no report generation/profiling capabilities**
- **No special compilation options are required**
- **Form of the time call**
  - **time myprogram inputfile outputfile**
- **Reported output**
  - **Bash shell**
    - **xxx real yyy user zzz sys**
  - **C shell**
    - **xxu yys zz vmem io pfw**

Call Trees



- Some observations
  - w and z are parents of x2
  - x1 and x2 are children of w
  - y1, y2, and y3 are children of x1 and descents of w

Outputs of gprof

- **A flat file**
  - **How much time was spent executing each function/subroutine**

- **A call graph**
  - **An extensive call graph that indicates how much time was spent based on parent child relationships**
  - **How much total time executing was based on calls from parents**
  - **Which calls to a procedure and which calls within a procedure take up time**

Preparing a File for Profiling

- **gcc -pg -o myfile myfi1e1.c myfile2.c …**
- **We note that -pg effects both the compiler and the linker**
  - **Compiler**
    - **Inserts code that keeps track of time and number of calls to each function**
  - **Linker**
    - **Causes necessary profiling routines to be included in the program**
- **Programs typically run 30% slower when profiling is enabled**
- **Don't forget to recompile without the -pg option before shipping it out the door**
- **Read the info documentation on gprof**

Creating a Profile With gprof

- **Step 1 compile program with the -pg option**
- **Step 2 run program to produce gmon.out files**
- **Step 3 run the following**
  - **gprof list-of-options executable-file stat-files > output**
    - **executable-file**
      - **Name of executable file that we are profiling**
      - **a.out by default**
    - **stat-files**
      - **gmon output files from various runs**
        - **If this is omitted then it generates the based on single file gmon.out**

gprof Options -I

- **-b**
  - **Express printing of profile table explanations**
- **-e name**
  - **Do not include the routine name in the call graph profile.**
  - **Do not include any of name’s descendents unless they are called bu other routines.**
  - **Time spent executing these programs is included in the totals.**
  - **Does not effect the flat profile**
  - **Can contain multiple routines**
    - **-e mysub1 -e mysub2**
- **-E name**
  - **Same as -e except time spent executing the omitted routines is not included in the totals**
- **-f name**
  - **Print the call grph profile for name and its descendents only**
  - **Note that the report includes time from all routines in its totals though**
  - **Can contain multiple routines**
  - **-f mysub1 -f mysub2**

gprof Options - II

- **-F name**
    - **Same as -f option except it includes on the times attributed to name and its descendents in the time calculation**
- **-s**
    - **Merge all listed stat files into a file known as gmon.sum**
- **-z**
    - **Generate a list of all routines that were not called**

Contains of a gprof Profile

- **Sections of a gprof Profile**
  - **Flat profile indicating the total time spent executing each routine**
  - **Call graph profile that analyzes execution time in terms of parents and children**
  - **List of programs and associated index numbers**

The Flat Profile

- **Amount of time spent in each function**
- **No information provided about the interactions between functions**
- **One line entry per function**
- **Sorted according to the routine that took the most time**

Contents of Each Entry in the Flat File

- **%time**
  - **Percentage of the program's total execution time spent executing this function. Note that it does not include time spent executing descendents of a particular function.**
- **cumulative seconds**
  - **Total time in seconds spent executing this function and all of the function that proceed it in the flat profile.**
- **self seconds**
  - **Amount of time spent executing this function. Note that it does not include its descendents.**
- **calls**
  - **Total number of calls that were made to this function.**
- **self ms/call**
  - **Average time in milliseconds that was spent executing this program each time it was called.**
- **total ms/call**
  - **Average time in milliseconds that was spent executing this program and each children each time it was called**
- **name**
  - **Name of the function**

Ways to Improve Performance

- **Improve routines that are utilized most so that they execute faster.**

- **Rewrite the code so that the most time consuming routines are called less frequently.**

- **Change routines to in-line routines**

Interpretation of the Call Graph Profile

- **When a functions self entry is higher than its children entry it is a good candidate for optimization**

- **When a functions children column is higher than its self column then focus attention on decreasing number of calls to children or in optimizing the children code**

- **When a function takes up a lot of time but you don't have control over optimizing it then try to reduce the number of times that it is called**

- **When an expensive child has a high value in the called column the try to reduce the number of times it is called. When it contains a low value in the called column then we should try to optimize its code.**