Измерение времени выполнения

Цели измерения времени

- Оценка (сравнение) платформ/библиотек/алгоритмов
- Анализ производительности
- Оптимизация
- Отладка ошибок, связанных со временем

Требования к процедуре измерения

- Повторяемость
- Проверяемость и переносимость
- Принцип невмешательства
- Приемлемый уровень точности
- Честность

Обзор способов измерения: критерии

- *Разрешение* выбранного способа измерения времени
- *Точность* измерений
- *Детализация* (англ. granularity) – часть кода, время выполнения которой измеряется
- *Сложность использования* (субъективный критерий)

Обзор способов измерения

Способ	**Разрешение**	**Точность**	**Детализация**	**Сложность**
Секундомер	0.01 с	0.5 с	Программа	Просто
date	0.02 с	0.2 с	Программа	Просто
time	0.02 с	0.2 с	Программа	Просто
gprof	10 мс	20 мс	Функции	Умеренно
“clock()”	15 – 30 мс	15 – 30 мс	Операторы	Умеренно

По [2]. Указанные значения приближены и субъективны.

date

```
$ cat get_time.sh
#!/bin/bash
date +"%T.%N" > log.txt
./app.exe >> log.txt
date +"%T.%N" >> log.txt

$ cat log.txt
17:35:44.993527905
17:35:44.999691474
```

time

```
$ time ./app.exe

real   0m0,006s
user   0m0,006s
sys    0m0,000s
```

time – утилита, возвращающая время выполнения команды.

- real – общее время от начала выполнения процесса и до его завершения.
- user – время, в течение которого процесс был в режиме пользователя.
- sys – время, в течение которого процесс был в режиме супервизора.

gprof

gprof - утилита анализа производительности

Позволяет анализировать

- количество вызовов функций, исполнения операторов и т.п.
- время выполнения функций, операторов и т.п.

Программа собирается с ключом -pg, который добавляет в программу специальный код. Во время выполнения программы различная статистика о ее выполнении накапливается в файле gmon.out. Затем этот файл анализируется утилитой gprof.

“clock()”

```
#include <time.h>
...
clock_t t_beg, t_end;
double total;
t_beg = clock();
  do stuff
t_end = clock();
total = (double) (t_end - t_beg) / CLOCKS_PER_SEC;
printf("Total = %f\n", total)
```

Если нужно большее разрешение?

- Организовать цикл
- Использовать способ измерения времени с большим разрешением :)

gettimeofday (POSIX)

```
#include <sys/time.h>
int gettimeofday(struct timeval *tp, void *tzp);
```

Получает текущее время, выраженное в секундах и микросекундах с 1 января 1970 года (Unix Epoch) и сохраняет его в структуру, на которую указывает tp. "Разрешение" системного таймера не указывается.

tzp - нулевой указатель (в противном случае поведение программы не определено).

Функция возвращает 0 и пока никаких кодов ошибок нет (?).

struct timeval (POSIX)

Заголовочный файл <sys/time.h> содержит определение структуры timeval, которая должна содержать по крайней мере два следующих поля:

```
time_t          tv_sec;     // секунды
suseconds_t     tv_usec;    // микросекунды
```

clock_gettime (POSIX)

```
#include <time.h>
int clock_gettime(clockid_t clockid, struct timespec *tp);
```

Возвращает время, указанное clockid.

clockid это идентификатор интересующих часов. Часы могут быть как относящимися ко всей системе, так и к отдельному процессу.

Все реализации должны поддерживать CLOCK_REALTIME.

clock_gettime (POSIX)

CLOCK_MONOTONIC - неустанавливаемые часы видимые во всей системе, которые идут "начиная с неопределенного момента в прошлом". На Linux это обычно число секунд, прошедших с загрузки ОС.

clock (POSIX)

```
#include <time.h>
clock_t clock(void);
```

Возвращает суммарное процессорное время, использованное программой. Чтобы узнать количество затраченных на выполнение секунд, необходимо разделить возвращенное значение на CLOCKS_PER_SEC.

Time Stamp Counter (TSC)

TSC – 64-х битный регистр, который появился во всех процессорах семейства x86, начиная с Pentium. Содержит число тактов с момента последнего сброса процессора.

Чаще всего используется:

- для измерения времени;
- для точного измерения временных интервалов;
- в антиотладочных целях;
- как источник энтропии для генераторов псевдослучайных чисел.

Time Stamp Counter (TSC)

```
#include <x86gprintrin.h>
unsigned long long __rdtsc(void);
```

Настоятельно рекомендуется использовать внутреннюю (англ. *intrinsic*) функцию, реализованную в gcc, а не самописную функцию на ассемблере.

Факторы, влияющие на измерения

- Всегда существуют другие процессы, соревнующиеся за ресурсы компьютера.
- Распределение ресурсов недетерминировано.
- Увеличение и ускорение частоты ЦП.

Обработка результатов измерений

- Среднее арифметическое значение: $t_{avg} = \frac{t_1 + t_2 + \cdots + t_n}{n}$
- Дисперсия: $s^2 = \frac{1}{n-1} \sum_1^n (t_i - t_{avg})^2$
- Стандартное отклонение: $s = \sqrt{s^2}$
- Стандартная ошибка: $StdErr = \frac{s}{\sqrt{n}}$
- Относительная стандартная ошибка среднего: $Rse = \frac{StdErr}{t_{avg}} * 100\%$

Обработка результатов измерений

Rse показывает на сколько близко вычисленное среднее время выполнения к истинному среднему времени выполнения (среднему генеральной совокупности).

В ряде работ [Курносов М.Г.] считается, что «На практике хорошая точность Rse <= 5%».

Организация повторов

Пример sort_5.c

Возможно, стоит подумать о повторах в течение заданного отрезка времени.

Можно использовать как количество повторов, так и интервал времени – что раньше кончится.

Учет влияния инфраструктуры

Пример sort_6.c

Условия выполнения измерений

- Выключите другие приложения.
- Используйте режим энергопотребления «высокая производительность».

Рекомендации

- Итерации для «прогрева» отбрасываются. «Прогрев» заканчивается, когда результаты перестают вести себя монотонно. (Эмпирическое правило.)
- Для первого запуска рекомендуется размер выборки от 15 до 30.
- Критерии остановки для реальных итераций могут быть основаны на стандартной ошибке.
- Коллективный опыт говорит, что для получения приемлемых результатов итерация должна длиться минимум 100 мс.

План исследования

1. Определение проблемы и целей.
2. Подбор правильных метрик.
3. Выбор подхода и инструментов.
4. Проведение эксперимента и получение результатов.
5. Анализ и формулирование выводов

Использованные материалы

1. Andrey Akinshin “Pro .NET Benchmarking: The Art of Performance Measurement”
2. David B. Stewart “Measuring Execution Time and Real-Time Perpformance”
3. Michael Kerrisk “The Linux Programming Interface”