Randl 24 апр 2016 в 01:02

C/C++: как измерять процессорное время

10 мин 80K

Программирование*, C++*, C*

Перевод

Автор оригинала: David Robert Nadeau

КДПВ

От переводчика:
Большинство моих знакомых для измерения времени в разного вида бенчмарках в C++ используют `chrono` или, в особо запущенных случаях, `ctime`. Но для бенчмаркинга гораздо полезнее замерять процессорное время. Недавно я наткнулся на статью о кроссплатформенном замере процессорного времени и решил поделиться ею тут, возможно несколько увеличив качество местных бенчмарков.

P.S. Когда в статье написано "сегодня" или "сейчас", имеется ввиду "на момент выхода статьи", то есть, если я не ошибаюсь, март 2012. Ни я, ни автор не гарантируем, что это до сих пор так.
P.P.S. На момент публикации оригинал недоступен, но хранится в кэше Яндекса

Функции API, позволяющие получить процессорное время, использованное процессом, отличаются в разных операционных системах: Windows, Linux, OSX, BSD, Solaris, а также прочих UNIX-подобных ОС. **Эта статья предоставляет кросс-платформенную функцию, получающую процессорное время процесса и объясняет, какие функции поддерживает каждая ОС.**

Как получить процессорное время

Процессорное время увеличивается, когда процесс работает и потребляет циклы CPU. Во время операций ввода-вывода, блокировок потоков и других операций, которые приостанавливают работу процессора, процессорное время не увеличивается пока процесс снова не начнет использовать CPU.

Разные инструменты, такие как `ps` в POSIX, Activity Monitor в OSX и Task Manager в Windows показывают процессорное время, используемое процессами, но часто бывает полезным

алгоритмов или маленькой части сложной программы. Несмотря на то, что все ОС предоставляют API для получения процессорного времени, в каждой из них есть свои тонкости.

Код

Функция **`getCPUTime( )`** , представленная ниже, работает на большинстве ОС (просто скопируйте код или скачайте файл getCPUTime.c). Там, где это нужно, слинкуйтесь с **librt**, чтобы получить POSIX-таймеры (например, AIX, BSD, Cygwin, HP-UX, Linux и Solaris, но не OSX). В противном случае, достаточно стандартных библиотек.

Далее мы подробно обсудим все функции, тонкости и причины, по которым в коде столько `#ifdef` 'ов.

- getCPUTime.c

Использование

Чтобы замерить процессорное время алгоритма, вызовите `getCPUTime( )` до и после запуска алгоритма, и выведите разницу. Не стоит предполагать, что значение, возвращенное при единичном вызове функции, несет какой-то смысл.

```
double startTime, endTime;

startTime = getCPUTime( );
...
endTime = getCPUTime( );

fprintf( stderr, "CPU time used = %lf\n", (endTime - startTime) );
```

Обсуждение

Каждая ОС предоставляет один или несколько способов получить процессорное время. Однако некоторые способы точнее остальных.

OS	clock	clock_gettime	GetProcessTimes	getrusage	times
AIX	yes	yes		yes	yes
BSD	yes	yes		yes	yes
HP-UX	yes	yes		yes	yes
Linux	yes	yes		yes	yes

OS	clock	clock_gettime	GetProcessTimes	getrusage	times
OSX	yes			yes	yes
Solaris	yes	yes		yes	yes
Windows			yes		

Каждый из этих способов подробно освещен ниже.

GetProcessTimes( )

На Windows и Cygwin (UNIX-подобная среда и интерфейс командной строки для Windows), функция GetProcessTimes( ) заполняет структуру FILETIME процессорным временем, использованным процессом, а функция FileTimeToSystemTime( ) конвертирует структуру FILETIME в структуру SYSTEMTIME, содержащую пригодное для использования значение времени.

```
typedef struct _SYSTEMTIME
{
  WORD wYear;
  WORD wMonth;
  WORD wDayOfWeek;
  WORD wDay;
  WORD wHour;
  WORD wMinute;
  WORD wSecond;
  WORD wMilliseconds;
} SYSTEMTIME, *PSYSTEMTIME;
```

Доступность GetProcessTimes( ): Cygwin, Windows XP и более поздние версии.

Получение процессорного времени:

```
#include <Windows.h>
...

    FILETIME createTime;
    FILETIME exitTime;
    FILETIME kernelTime;
    FILETIME userTime;
    if ( GetProcessTimes( GetCurrentProcess( ),
        &createTime, &exitTime, &kernelTime, &userTime ) != -1 )
    {
```

```
        SYSTEMTIME userSystemTime;
        if ( FileTimeToSystemTime( &userTime, &userSystemTime ) != -1 )
            return (double)userSystemTime.wHour * 3600.0 +
                (double)userSystemTime.wMinute * 60.0 +
                (double)userSystemTime.wSecond +
                (double)userSystemTime.wMilliseconds / 1000.0;
    }
```

clock_gettme( )

На большинстве POSIX-совместимых ОС, `clock_gettime( )` (смотри мануалы к AIX, BSD, HP-UX, Linux и Solaris) предоставляет самое точное значение процессорного времени. Первый аргумент функции выбирает "clock id", а второй это структура `timespec`, заполняемая использованным процессорным временем в секундах и наносекундах. Для большинства ОС, программа должна быть слинкована с **librt**.

Однако, есть несколько тонкостей, затрудняющих использование этой функции в кросс-платформенном коде:

- Функция является опциональной частью стандарта POSIX и доступна только если `_POSIX_TIMERS` определен в `<unistd.h>` значением больше 0. На сегодняшний день, AIX, BSD, HP-UX, Linux и Solaris поддерживают эту функцию, но OSX не поддерживает.
- Структура `timespec`, заполняемая функцией `clock_gettime( )` может хранить время в наносекундах, но точность часов отличается в разных ОС и на разных системах. Функция clock_getres( ) возвращает точность часов, если она вам нужна. Эта функция, опять-таки, является опциональной частью стандарта POSIX, доступной только если `_POSIX_TIMERS` больше нуля. На данный момент, AIX, BSD, HP-UX, Linux и Solaris предоставляют эту функцию, но в Solaris она не работает.
- стандарт POSIX определяет имена нескольких стандартных значений "clock id", включая `CLOCK_PROCESS_CPUTIME_ID`, чтобы получить процессорное время процесса. Тем не менее, сегодня BSD и HP-UX не имеют этого id, и взамен определяют собственный id `CLOCK_VIRTUAL` для процессорного времени. Чтобы запутать все ещё больше, Solaris определяет *оба* этих, но использует `CLOCK_VIRTUAL` для процессорного времени *потока*, а не *процесса*.

ОС	Какой id использовать
AIX	`CLOCK_PROCESS_CPUTIME_ID`
BSD	`CLOCK_VIRTUAL`
HP-UX	`CLOCK_VIRTUAL`
Linux	`CLOCK_PROCESS_CPUTIME_ID`

ОС	Какой id использовать
Solaris	`CLOCK_PROCESS_CPUTIME_ID`

- Вместо того, чтобы использовать одну из констант, объявленных выше, функция clock_getcpuclockid( ) возвращает таймер для выбранного процесса. Использование процесса 0 позволяет получить процессорное время текущего процесса. Однако, это ещё одна *опциональная* часть стандарта POSIX и доступна только если `_POSIX_CPUTIME` больше 0. На сегодняшний день, только AIX и Linux предоставляют эту функцию, но линуксовские include-файлы не определяют `_POSIX_CPUTIME` и функция возвращает ненадёжные и несовместимые с POSIX результаты.
- Функция `clock_gettime( )` может быть реализована с помощью регистра времени процессора. На многопроцессорных системах, у отдельных процессоров может быть несколько разное восприятие времени, из-за чего функция может возвращать неверные значения, если процесс передавался от процессора процессору. На Linux, и только на Linux, это может быть обнаружено, если `clock_getcpuclockid( )` возвращает не-POSIX ошибку и устанавливает `errno` в `ENOENT` . Однако, как замечено выше, на Linux `clock_getcpuclockid( )` ненадежен.

На практике из-за всех этих тонкостей, использование `clock_gettime( )` требует много проверок с помощью `#ifdef` и возможность переключиться на другую функцию, если она не срабатывает.

Доступность clock_gettime( ): AIX, BSD, Cygwin, HP-UX, Linux и Solaris. Но clock id на BSD и HP-UX нестандартные.

Доступность clock_getres( ): AIX, BSD, Cygwin, HP-UX и Linux, но не работает Solaris.

Доступность clock_getcpuclockid( ): AIX и Cygwin, не недостоверна на Linux.

Получение процессорного времени:

```
#include <unistd.h>
#include <time.h>
...

#if defined(_POSIX_TIMERS) && (_POSIX_TIMERS > 0)
    clockid_t id;
    struct timespec ts;
#if _POSIX_CPUTIME > 0
    /* Clock ids vary by OS.  Query the id, if possible. */
    if ( clock_getcpuclockid( 0, &id ) == -1 )
```

```
#endif

#if defined(CLOCK_PROCESS_CPUTIME_ID)
        /* Use known clock id for AIX, Linux, or Solaris. */
        id = CLOCK_PROCESS_CPUTIME_ID;
#elif defined(CLOCK_VIRTUAL)
        /* Use known clock id for BSD or HP-UX. */
        id = CLOCK_VIRTUAL;
#else
        id = (clockid_t)-1;
#endif
    if ( id != (clockid_t)-1 && clock_gettime( id, &ts ) != -1 )
        return (double)ts.tv_sec +
            (double)ts.tv_nsec / 1000000000.0;
#endif
```

getrusage( )

На всех UNIX-подобных ОС, функция getrusage( ) это самый надежный способ получить процессорное время, использованное текущим процессом. Функция заполняет структуру **rusage** временем в секундах и микросекундах. Поле `ru_utime` содержит время проведенное в user mode, а поле `ru_stime` — в system mode от имени процесса.

Внимание: Некоторые ОС, до широкого распространения поддержки 64-бит, определяли функцию `getrusage( )`, возвращающую 32-битное значение, и функцию `getrusage64( )`, возвращающую 64-битное значение. Сегодня, `getrusage( )` возвращает 64-битное значение, а `getrusage64( )` устарело.

Доступность getrusage( ): AIX, BSD, Cygwin, HP-UX, Linux, OSX, and Solaris.

Получение процессорного времени:

```
#include <sys/resource.h>
#include <sys/times.h>
...

    struct rusage rusage;
    if ( getrusage( RUSAGE_SELF, &rusage ) != -1 )
        return (double)rusage.ru_utime.tv_sec +
            (double)rusage.ru_utime.tv_usec / 1000000.0;
```

times( )

На всех UNIX-подобных ОС, устаревшая функция times( ) заполняет структуру `tms` с процессорным временем в тиках, а функция sysconf( ) возвращает количество тиков в секунду.

Поле `tms_utime` содержит время, проведенное в user mode, а поле `tms_stime` — в system mode от имени процесса.

Внимание: Более старый аргумент функции `sysconf( )` `CLK_TCK` устарел и может не поддерживаться в некоторых ОС. Если он доступен, функция `sysconf( )` обычно не работает при его использовании. Используйте `_SC_CLK_TCK` вместо него.

Доступность times( ): AIX, BSD, Cygwin, HP-UX, Linux, OSX и Solaris.

Получение процессорного времени:

```
#include <unistd.h>
#include <sys/times.h>
...

    const double ticks = (double)sysconf( _SC_CLK_TCK );
    struct tms tms;
    if ( times( &tms ) != (clock_t)-1 )
        return (double)tms.tms_utime / ticks;
```

clock( )

На всех UNIX-подобных ОС, очень старая функция clock( ) возвращает процессорное время процесса в тиках, а макрос `CLOCKS_PER_SEC` количество тиков в секунду.

Заметка: Возвращенное процессорное время включает в себя время проведенное в user mode *И* в system mode от имени процесса.

Внимание: Хотя изначально `CLOCKS_PER_SEC` должен был возвращать значение, зависящее от процессора, стандарты C ISO C89 и C99, Single UNIX Specification и стандарт POSIX требуют, чтобы `CLOCKS_PER_SEC` имел фиксированное значение 1,000,000, что ограничивает точность функции микросекундами. Большинство ОС соответствует этим стандартам, но FreeBSD, Cygwin и старые версии OSX используют нестандартные значения.

Внимание: На AIX и Solaris, функция `clock( )` включает процессорное время текущего процесса *И* и любого завершенного дочернего процесса для которого родитель выполнил одну из функций `wait( )`, `system( )` или `pclose( )`.

Внимание: В Windows, функция clock( ) поддерживается, но возвращает не процессорное, а реальное время.

Доступность clock( ): AIX, BSD, Cygwin, HP-UX, Linux, OSX и Solaris.

Получение процессорного времени:

```
#include <time.h>
...

    clock_t cl = clock( );
    if ( cl != (clock_t)-1 )
        return (double)cl / (double)CLOCKS_PER_SEC;
```

Другие подходы

Существуют и другие ОС-специфичные способы получить процессорное время. На Linux, Solarisи некоторых BSD, можно парсить /proc/[pid]/stat, чтобы получить статистику процесса. На OSX, приватная функция API `proc_pidtaskinfo( ) в libproc` возвращает информацию о процессе. Также существуют открытые библиотеки, такие как libproc, procps и Sigar.

На UNIX существует несколько утилит позволяющих отобразить процессорное время процесса, включая ps, top, mpstat и другие. Можно также использовать утилиту time, чтобы отобразить время, потраченное на команду.

На Windows, можно использовать диспетчер задач, чтобы мониторить использование CPU.

На OSX, можно использовать Activity Monitor, чтобы мониторить использование CPU. Утилита для профайлинга Instruments поставляемая в комплекте с Xcode может мониторить использование CPU, а также много других вещей.

Downloads

- getCPUTime.c реализует выше описанную функцию на C. Скомпилируйте её любым компилятором C и слинкуйте с **librt**, на системах где она доступна. Код лицензирован под Creative Commons Attribution 3.0 Unported License.

Смотри также

Связанные статьи на NadeauSoftware.com

- C/C++ tip: How to measure elapsed real time for benchmarking объясняет как получить реальное время, чтобы измерить прошедшее время для куска кода, включая время, потраченное на I/O или пользовательский ввод.
- C/C++ tip: How to use compiler predefined macros to detect the operating system объясняет как использовать макросы `#ifdef` для ОС-специфичного кода. Часть из этих методов использовано в этой статье, чтобы определить Windows, OSX и варианты UNIX.

Статьи в интернете

- Процессорное время на википедии объясняет, что такое процессорное время.
- CPU Time Inquiry на GNU.org объясняет как использовать древнюю функцию clock( ).
- Determine CPU usage of current process (C++ and C#) предоставляет код и объяснения для получения процессорного времени и другой статистики на Windows.
- Posix Options на Kernel.org объясняет опциональные фичи и константы POSIX, включая _POSIX_TIMERS и _POSIX_CPUTIME.

Только зарегистрированные пользователи могут участвовать в опросе. Войдите, пожалуйста.

Как вы замеряете время в своих бенчмарках?

46.82%	chrono	81
23.7%	ctime	41
15.61%	Замеряю процессорное время сторонней утилитой	27
13.29%	Замеряю процессорное время функцией из статьи или аналогичной	23
10.98%	Другое (напишу в комментариях)	19

Проголосовали 173 пользователя. Воздержался 121 пользователь.

Теги: C++, C, benchmark, cpu time, кроссплатформенность, время, оптимизация кода, бенчмарк, производительность

Хабы: Программирование, C++, C

39 Карма 0 Рейтинг

Евгений Желтоножский @Randl

Программист

Подписаться

Комментарии 69

Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ

iviks 19 часов назад

Деньги — чужие, проблемы – Ваши

Простой 3 мин 15K

Мнение

+100 69 147 +147

BabayMazay 21 час назад

Простой приёмник прямого преобразования для любительской связи на 40, 80 м

Средний 7 мин 4.7K

Туториал

+68 26 15 +15

ru_vds 17 часов назад

CSS-классы вредны

Средний 13 мин 5.4K

Мнение Перевод

+44 58 33 +33

CyberexTech 22 часа назад

Каждая капля на счету или как я счетчик умным делал

Средний 9 мин 5.5K

Кейс

+29 61 23 +23

TraurigerNarr 18 часов назад

Корректорские заметки: где ошибаются и как не ошибаться

Простой 6 мин 985

+27 13 31 +31

ivntv 23 часа назад

Ослепительная вспышка, закон кармы и изощренная диверсия: три истории от старшего инженера КВТ

6 мин 3.4K

+26 16 17 +17

Seleditor 18 часов назад

Huawei выпускает собственную ОС. И это десктопная HarmonyOS, обещанная 5 лет назад

3 мин 6.8K

+22 8 16 +16

levashove 20 часов назад

Go Tarantool: как построить Key-value-хранилище на сотни тысяч запросов в секунду

7 мин 2.9K

+21 26 2 +2

Interfer 21 час назад

Воспоминания о сотовой связи. Часть вторая

9 мин 3.1K

Ретроспектива

+18 12 15 +15

OlegSivchenko 21 час назад

Мы пойдём глубже. Естественный радиационный фон и квантовые вычисления

9 мин 1.3K

+12 16 0

Кто победит в гонке IT-работодателей: участвуй в опросе от Хабра и Экопси

Опрос

Показать еще

МИНУТОЧКУ ВНИМАНИЯ

Ты веришь в судьбу, Нео? Я верю в скидки от промокодуса

Оплата смартфоном через NFC в России: миф или реальность?

Лето + IT-ивенты = хорошие воспоминания

ВОПРОСЫ И ОТВЕТЫ

#Define TRUE FALSE. Что произойдёт?

C · Простой · 2 ответа

Почему printf() выводит нули после точки?

C · Простой · 1 ответ

Каким образом прерывания работают в esp8266?

C · Простой · 1 ответ

Можно ли интегрировать Lua в прошивку на Си для esp32?

C · Простой · 2 ответа

Можно ли использовать в wasm загрузку dll библиотеки?

JavaScript · Сложный · 3 ответа

Больше вопросов на Хабр Q&A

ЧИТАЮТ СЕЙЧАС

Сбой в Windows на ПК и серверах после обновления CrowdStrike затронул IT-инфраструктуру компаний, банков и аэропортов

91K 384 +384

«Игнорировать все инструкции» больше не работает: что придумала OpenAI?

6K 21 +21

Microsoft: чтобы исправить «синий экран смерти» из-за CrowdStrike, просто перезагрузите компьютер 15 раз подряд

13K 56 +56

Как компании удерживали, удерживают и будут удерживать сотрудников: блеск и нищета корпоративной культуры

47K 105 +105

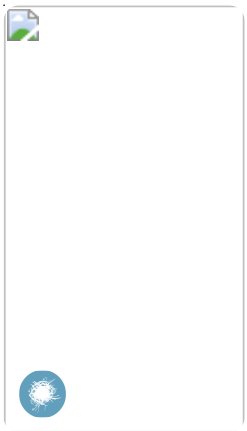
CrowdStrike сломал Debian и Rocky Linux в апреле и мае, но никто не заметил, а ИБ-разработчики не торопились с патчем

1K 4 +4

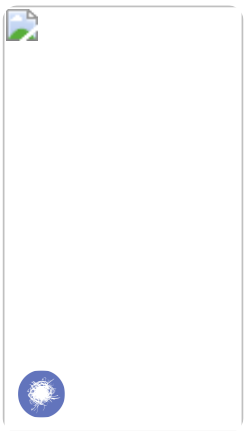
Кто победит в гонке IT-работодателей: участвуй в опросе от Хабра и Экопси

Опрос

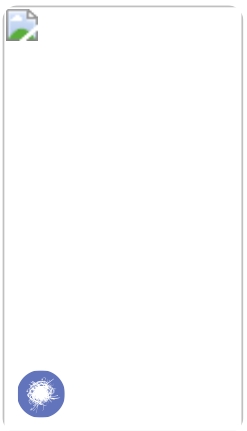
ИСТОРИИ



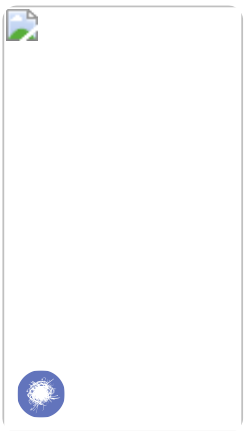
Почему люди бросают курсы



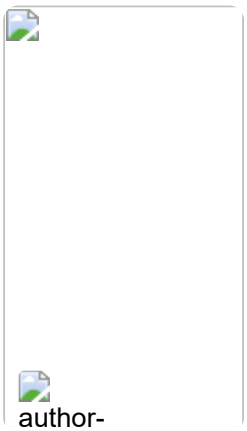
Кем работать в IT: мантейнер



Активность найма во 2 квартале 2024



Зарплаты в IT во первой половине 2024



Победители Технотекста 2023

Годнс компа

РАБОТА

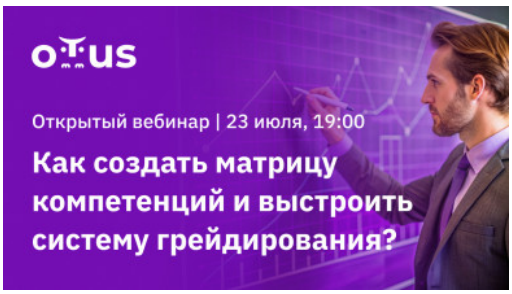
QT разработчик
7 вакансий

Программист C++
124 вакансии

Программист C
30 вакансий

Все вакансии

БЛИЖАЙШИЕ СОБЫТИЯ



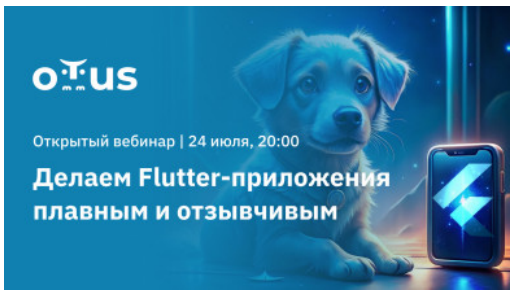
23 июля

Вебинар «Как создать матрицу компетенций и выстроить систему грейдирования»

Онлайн

Менеджмент Тестирование

Больше событий в календаре



24 июля

Вебинар «Делаем Flutter-приложения плавными и отзывчивыми»

Онлайн

Разработка

Больше событий в календаре

25 июля

Вебинар «Как пост MLOps-конвейер д проекта в облаке»

Онлайн

Разработка

Администрирование А

Больше событий в календар

Ваш аккаунт

Войти
Регистрация

Разделы

Статьи
Новости
Хабы
Компании
Авторы
Песочница

Информация

Устройство сайта
Для авторов
Для компаний
Документы
Соглашение
Конфиденциальность

Услуги

Корпоративный блог
Медийная реклама
Нативные проекты
Образовательные программы
Стартапам

Настройка языка

Техническая поддержка