МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«ОПРЕДЕЛЕНИЕ ВРЕМЕНИ РАБОТЫ ПРИКЛАДНЫХ ПРОГРАММ»

студента 2 курса, 21211 группы

Михалёв Макар Андреевич

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Кудинов Антон Юрьевич

Цели работы

1 Изучение методики измерения времени работы подпрограммы.

2 Изучение приемов повышения точности измерения времени работы
подпрограммы.

3 Изучение способов измерения времени работы подпрограммы.

4 Измерение времени работы подпрограммы в прикладной программе.

Задание к лабороторной работе

1 Написать программу на языке C или C++, которая реализует выбранный алгоритм из задания.

2 Проверить правильность работы программы на нескольких тестовых наборах входных данных.

3 Выбрать значение параметра N таким, чтобы время

работы программы было порядка 15 секунд.

4 По приведенной методике определить время работы подпрограммы тестовой программы с относительной погрешностью не более 1%.

Описание работы

Было представлен несколько методов решений проблемы измерения времени. Первая из которых воспользовался:

Утилита `time` - измеряет время работы приложения.

Библиотечная функция `clock_gettime` получает значения системного таймера

Библиотечная функция times определяет время работы процесса в многозадачной операционной системе.

Машинная команда rdtsc, являющаяся по своей сути счётчиком тактов процессора. Этот метод является максимально точным, определяя время работы с точностью до такта процессора.

Описание работы

В ходе работы были изучены и испробованы таймеры также был реализован алгоритм вычисления числа Пи с помощью разложения в ряд (ряд
Грегори-Лейбница) по формуле Лейбница N первых членов ряда.Из эксперимента было замечено, что наименьшей точность была у утилиты time. Стоит отметить, что не был подключен сброс буфера отложенной записи на диск так как утилита считает время всей программы и это бы

сильно изменило показатели. Так же были испробованы clock_gettime первый плюс измерение времени участка, но так же возможна не точность. И последняя испробованная библиотечная функция times, плюс измерения одного процессы. Все эти таймеры имеют погрешность и неточность измерения, но измеряют как малыш так и большие участки кода и весь код .Машиная команда не была испробована так как идет привязка к архитектуре x86, но было изучено что обладает хореей точностью относительно исправных таймеров, еще один минус измеряет малые участки кода

Описание *методики измерения времени*

Утилита `time` выдаёт следующие временные характеристики работы программы:

real – общее время работы программы согласно системному таймеру, полное процеснное время

user – время, которое работал пользовательский процесс (кроме времени работы других процессов)

sys – время, затраченное на выполнение системных вызовов программы. Время служебных операций который инициировал код.

Точность: определяется точностью системного таймера и точностью измерения времени работы процесса (см. описание соответствующих таймеров ниже). Достоинство: готовая утилита, не требуется вносить изменения в программу. Недостаток: измеряется только время работы всей программы, нет возможности измерить время работы отдельных её частей.

Функция `clock_gettime` с параметром `CLOCK_MONOTONIC_RAW` сохраняет значение системного таймера в структуру `struct timespec`. Структура состоит из двух полей: `tv_sec` и `tv_nsec`, задающих количество секунд и наносекунд (10^{-9} сек.), прошедших с некоторого неспецифицированного момента времени в прошлом.

Достоинство: переносимость – вне зависимости от аппаратного обеспечения функция доступна пользователю, т.к. реализуется ОС.

Недостатки: относительно низкая точность (обычно ниже, чем у счётчика тактов, но выше, чем у функции `times`), и измеренный интервал включает время работы других процессов, которые работали на процессоре в измеряемый период.

Функция измерения времени работы процесса `times` позволяет определить время работы данного процесса в многозадачной операционной системе, где каждый процессор (или ядро) выполняет несколько процессов (программ) в режиме разделения времени. Этот показатель особенно важен, когда процессор сильно загружен другими процессами. В этом случае показания, например, системного таймера могут быть очень далеки от действительного времени работы программы. Процессор переключается между процессами с некоторой периодичностью Функция `times` позволяет определить, сколько таких квантов времени проработал наш процесс. Если перевести количество этих квантов во время, то можно определить, какое время работал процесс.

Точность: зависит от кванта планировщика процессов, обычно 10 мс (10^{-2} с).

Достоинство: Из измеряемого времени исключается время, которое работали другие процессы, это обеспечивает более точное измерение времени работы программы на сильно загруженных процессорах по сравнению с другими способами.

Недостаток: относительно низкая точность, определяемая квантом времени переключения процессов.

Результаты работы

	1	2	3	4	5	6	7	8	9
clock_gettime	15.036211 sec	15.094580 sec	15.060714 sec	15.080431 sec	15.053255 sec	15.030874 sec	15.072458 sec	15.077995 sec	15.167649 sec

time	15,380 sec	15,317 sec	15,363 sec	15,315 sec	15,061 sec	15,299 sec	15,250 sec	15,342 sec	15,355 sec
Times	14.960000 sec	14.990000 sec	14.970000 sec	14.970000 sec.	14.980000 sec	14.960000 sec	14.970000 sec	14.980000 sec	14.990000 se

Листинг программы

```
#include <iostream>
#include <ctime>

inline double EvenNumber(long long i)
{
    return (i % 2 == 0) ? 1.0 : -1.0;
}

double  SerExpanGregoryLeibniz()
{
    long long N = 16000000000;
    double res = 0, signDef = 1.0;
    for(long long i = 0; i < N; ++i)
    {
        res += (4.0 * signDef / (2.0 * (i) + 1.0));
        signDef *= (-1);
    }
    return res;
}

int main(int argc, const char * argv[]) {
    std::system("sync");

    struct timespec start, end;

    clock_gettime(CLOCK_MONOTONIC_RAW, &start);


    SerExpanGregoryLeibniz();
```

```
    clock_gettime(CLOCK_MONOTONIC_RAW, &end);

    printf("Time taken: %lf sec.\n", end.tv_sec-start.tv_sec +
0.000000001*(end.tv_nsec - start.tv_nsec));

    return 0;
}
```

```
#include <iostream>
#include <sys/times.h>
#include <unistd.h>
inline double EvenNumber(long long i)
{
    return (i % 2 == 0) ? 1.0 : -1.0;
}

double  SerExpanGregoryLeibniz()
{
    long long N = 16000000000;
    double res = 0, signDef = 1.0;
    for(long long i = 0; i < N; ++i)
    {
        res += (4.0 * signDef / (2.0 * (i) + 1.0));
        signDef *= (-1);
    }
    return res;
}

int main(int argc, const char * argv[]) {
    std::system("sync");
    struct tms start, end;
    long clocks_per_sec = sysconf(_SC_CLK_TCK);
    long clocks;
    times(&start);
    SerExpanGregoryLeibniz();

   times(&end);
   clocks = end.tms_utime - start.tms_utime;
   printf("Time taken: %lf sec.\n",
   (double)clocks / clocks_per_sec);
    return 0;
}
```

Вывод:

В результате эксперимента были подтверждены плюсы и минусы каждого таймера.