**Смирнов Иван ИУ7-22Б - 2023г.**

**Отчет**

**Задание №4**

**Исследование характеристик программного обеспечения**

*Целью* работы является изучение расположения в памяти локальных переменных и представления структур.  
**Задание №1**

По условию задания был проведен эксперимент замера среднего времени выполнения функции nanosleep для задержки в 1с, 100мс, 50мс, 10мс разными методами (gettimeofday, clock\_gettime, clock, \_\_rdtsc). Для этого было создано несколько программ с разными методами замера времени (см. папку /task\_4/1):

Вариант 1 – **gettimeofday (main-1.c)**

#include <stdio.h>

#include <time.h>

#include <sys/time.h>

#define OK 0

struct timespec

{

   time\_t tv\_sec;

   long tv\_nsec;

};

int nanosleep(const struct timespec \*req, struct timespec \*rem);

// Замерительный метод - gettimeofday

int main (void)

{

   struct timespec tw = {0,10\*1e+6};

   struct timespec tr;

   struct timeval current\_time;

   unsigned long long beg, end;

   gettimeofday(&current\_time, NULL);

   beg = current\_time.tv\_sec \* 1000ULL + current\_time.tv\_usec / 1000ULL;

   nanosleep(&tw, &tr);

   gettimeofday(&current\_time, NULL);

   end = current\_time.tv\_sec \* 1000ULL + current\_time.tv\_usec / 1000ULL;

   printf("%llu\n", end-beg);

   return OK;

}

Вариант 2 – **clock\_gettime (main-2.c)**

#define \_POSIX\_C\_SOURCE 199309L

#include <time.h>

#include <stdio.h>

#include <unistd.h>

#define OK 0

// Замерительный метод - clock\_gettime

int main (void)

{

   struct timespec tw = {0,10\*1e+6};

   struct timespec tr;

   struct timespec start, ending;

   unsigned long long beg, end;

   clock\_gettime( CLOCK\_REALTIME, &start );

   beg = start.tv\_sec \* 1000ULL + start.tv\_nsec / 1000ULL;

   nanosleep(&tw, &tr);

   clock\_gettime( CLOCK\_REALTIME, &ending );

   end = ending.tv\_sec \* 1000ULL + ending.tv\_nsec / 1000ULL;

   printf("%llu\n", end-beg);

   return OK;

}

Вариант 3 – **clock (main-3.c)**

#include <time.h>

#include <stdio.h>

#include <unistd.h>

#define OK 0

struct timespec

{

   time\_t tv\_sec;

   long tv\_nsec;

};

int nanosleep(const struct timespec \*req, struct timespec \*rem);

// Замерительный метод - clock()

int main (void)

{

   struct timespec tw = {0,10\*1e+6};

   struct timespec tr;

   double time\_spent = 0;

   clock\_t begin = clock();

   nanosleep(&tw, &tr);

   clock\_t end = clock();

   time\_spent += (double)(end - begin) \* 1000ULL / CLOCKS\_PER\_SEC;

   printf("%f\n", time\_spent);

   return OK;

}

Вариант 4 – **\_\_rdtsc (main-4.c)**

#include <time.h>

#include <stdio.h>

#include <unistd.h>

#include <x86gprintrin.h>

#define OK 0

struct timespec

{

   time\_t tv\_sec;

   long tv\_nsec;

};

int nanosleep(const struct timespec \*req, struct timespec \*rem);

// Замерительный метод - \_\_rdtsc()

int main (void)

{

   struct timespec tw = {0,10\*1e+6};

   struct timespec tr;

   unsigned long long t1 = \_\_rdtsc();

   nanosleep(&tw, &tr);

   unsigned long long t2 = \_\_rdtsc();

   printf("%llu\n", (t2 - t1)/CLOCKS\_PER\_SEC);

   return OK;

}

Так же в рабочей папке присутствуют скрипты для удобства работы с измерениями.

***build\_release.sh ($key) –*** скрипт сборки программы с ключом (ключ указывает на номер собираемой программы, например, команда **./build\_release.sh 3** соберет программу main-3.c).

***dataset.sh ($key) –*** скрипт, который выполняет 20 тестовых измерений времени у программы (номер программы определяется ключом). Результаты измерений помещаются в текстовые файлы вида **./dataset/t\_$key\_$i.txt**, где i – номер измерения (теста).

***get\_avg.sh ($key) –*** скрипт, который на основе данных, полученных из **dataset.sh**, считает среднее арифметическое измерений и записывает его в файл **./avg/t\_$key.txt**.

Вспомогательные скрипты **check\_scripts.sh** (проверка shellcheck всех скриптов), **chmod.sh** (выдача права на изменение для всех скриптов) и **clean.sh** (очистка временных файлов, а также текстовых файлов из каталогов ./dataset и ./avg) были взяты из лабораторных работ курса “Программирование на языке Си”.

Ниже приведена таблица измерений (время измеряется в мс):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Реальное время | gettimeofday | clock\_gettime | clock | \_\_rdtsc |
| 1000 | 1000 | 14060 | 0.8 | 2501 |
| 100 | 102 | 7377 | 0.05 | 255 |
| 50 | 53 | 1035 | 0.07 | 126 |
| 10 | 12 | 13625 | 0.02 | 31 |

Как видно из таблицы наиболее точный метод – **gettimeofday.** Именно этот метод был использован при решении задания №2.

Метод \_\_rdtsc имеет погрешность результата почти в 1.5 раза во всех рассматриваемых случаях. То есть присутствует закономерность, однако смещение в 1.5 раза - достаточно большое.

Методы clock\_gettime и clock (из данных таблицы) являются самыми нестабильными. На разных машинах (Ubuntu 22/ Windows 11) функции выдают разные результаты. Clock() сообщает, сколько процессорного времени используется; процессорное время в свою очередь зависит от количества активных потоков (которое в разные состояния может быть разным). Из-за этого пользователь получает не тот результат, который хотел бы увидеть.

**Задание №2**