Отчет о временном профилировщике

Постановка задачи:

Необходимо создать потокобезопасный временной профилировщик с маркросом, который можно будет включать и выключать во время компиляции и выполнения. Профилировщик должен быть максимально оптимизирован.

Метод решения:

Решение основано на принципе ***RAII***. Создается массив с фиксированным размером, состоящий из структур, каждая из которых содержит информацию об отдельной функции: сколько раз она выполнялась, накопленное время выполнения, накопленное квадратное время выполнения и имя функции. Для получения имени функции используется макрос ***__func__***. Правильная ячейка для каждой функции вычисляется, используя тот факт, что статические переменные инициализируются только один раз. Для учета выполнения функций в разных потоках используются атомарные переменные внутри информационной структуры, а также упорядочивание памяти для повышения производительности.
Результаты вычислений записываются в файл при завершении программы, используя факт удаления статических глобальных переменных в конце программы.

Использование:

Для измерения времени выполнения функций используется макрос ***PROFILE_SCOPE()***, который необходимо поместить внутри функции.
Для отключения профилировщика во время выполнения программы используется макрос ***PROFILE_OFF()***, а для включения - макрос ***PROFILE_ON()***. По умолчанию профилировщик включен.
Для включения профилирования используется флаг времени компиляции ***PROFILE_ENABLED***.

Результаты тестирования:

A. QueueA / QueueL

1. Добавление некоторого количества элементов, а затем их удаление

Regular	Compile time off	Run time off	Run time on
QueueA	38204 μs	86955 μs	423440 μs
QueueL	54008 μs	102272 μs	436434 μs

QueueA	Time
Push	0.01698 ± 9.11891 μs
Pop	0.00065 ± 0.19121 μs

QueueL	Time
Push	0.00219 ± 0.247421 μs
Pop	0.00196 ± 0.33401 μs

2. Добавление элемента и его немедленное удаление

Push-Pop	Compile time off	Run time off	Run time on
QueueA	18556 μs	68799 μs	457146 μs
QueueL	41549 μs	99336 μs	465765 μs

QueueA	Time
Push	0.0015312 ± 0.23393 μs
Pop	0.00103 ± 0.16257 μs

QueueL	Time
Push	0.00197 ± 0.4650 μs
Pop	0.0021 ± 0.4580 μs

3. Добавление некоторого количества элементов, затем их удаление, снова добавление некоторого количества и затем их удаление

Push_Pop-Push_Pop	Compile time off	Run time off	Run time on
QueueA	14880 μs	42430 μs	223269 μs
QueueL	28136 μs	57071 μs	244224 μs

QueueA	Time
Push	0.00962 ± 3.493193 μs
Pop	0.00079 ± 0.15531 μs

QueueL	Time
Push	0.0019 ± 0.18801 μs
Pop	0.0018 ± 0.27743 μs

B. Threads**.**

FunctionA - функция, которая выполняется во многих потоках одновременно и просто спит в течение 20ms
FunctionB - функция, которая выполняется во многих потоках одновременно и просто спит в течение 25ms
FucntionC - функция, которая выполняется во многих потоках одновременно и просто спит в течение 30ms

Push_Pop-Push_Pop	Compile time off	Run time off	Run time on
Thread	38319 μs	38374 μs	37926 μs

Thread	Time
FunctionA	20.067 ± 0.0274 ms
FunctionB	25.061 ± 0.0169 ms
FunctionC	30.062 ± 0.0147 ms

Воспроизведение результатов тестирования:

Запустите *prj.test/timerprofiler_test.cpp с нужным тестом в main*, результаты можно получить из *bin.rel/TimeProfiler.txt*

Вывод*:*
Временной профилировщик продемонстрировал эффективную работу при анализе выполнения программы. Хотя некоторые результаты тестирования выглядели нереалистично из-за отсутствия ожидаемых значений или стандартных отклонений для некоторых операций, например, амортизированных, общее влияние на функциональность программы было минимальным. Несмотря на то, что профилировщик времени вызвал небольшое увеличение времени работы программы, он практически не повлиял на время выполнения отдельных функций. Это наблюдение особенно ярко проявилось в результатах тестирования многопоточных функций.