Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

Факультет программной инженерии и компьютерной техники

Отчет по лабораторной работе №1

по дисциплине

«Операционные системы»

Вариант: Windows, CreateProcessInternal, io-lat-read, factorize, 1KB

Выполнил: студент группы P3309

Черноморов Кирилл Александрович

Преподаватель:

Осипов Святослав Владимирович

Санкт-Петербург
2024 г.

Часть 1. Запуск программ

Необходимо реализовать собственную оболочку командной строки - shell. Выбор ОС для реализации производится на усмотрение студента. Shell должен предоставлять пользователю возможность запускать программы на компьютере с переданными аргументами командной строки и после завершения программы показывать реальное время ее работы (подсчитать самостоятельно как «время завершения» – «время запуска»).

Часть 2. Мониторинг и профилирование

Разработать комплекс программ-нагрузчиков по варианту, заданному преподавателем. Каждый нагрузчик должен, как минимум, принимать параметр, который определяет количество повторений для алгоритма, указанного в задании. Программы должны нагружать вычислительную систему, дисковую подсистему или обе подсистемы сразу. Необходимо скомпилировать их без опций оптимизации компилятора.

Перед запуском нагрузчика, попробуйте оценить время работы вашей программы или ее результаты (если по варианту вам досталось измерение чего либо). Постарайтесь обосновать свои предположения. Предположение можно сделать, основываясь на свой опыт, знания ОС и характеристики используемого аппаратного обеспечения.

1. Запустите программу-нагрузчик и зафиксируйте метрики ее работы с помощью инструментов для профилирования. Сравните полученные результаты с ожидаемыми. Постарайтесь найти объяснение наблюдаемому.
2. Определите количество нагрузчиков, которое эффективно нагружает все ядра процессора на вашей системе. Как распределяются времена USER%, SYS%, WAIT%, а также реальное время выполнения нагрузчика, какое количество переключений контекста (вынужденных и невынужденных) происходит при этом?
3. Увеличьте количество нагрузчиков вдвое, втрое, вчетверо. Как изменились времена, указанные на предыдущем шаге? Как ведет себя ваша система?
4. Объедините программы-нагрузчики в одну, реализованную при помощи потоков выполнения, чтобы один нагрузчик эффективно нагружал все ядра вашей системы. Как изменились времена для того же объема вычислений? Запустите одну, две, три таких программы.
5. Добавьте опции агрессивной оптимизации для компилятора. Как изменились времена? На сколько сократилось реальное время исполнения программы нагрузчика?

Реализация

см. весь код GitHub - https://github.com/Gallade901/OS-1

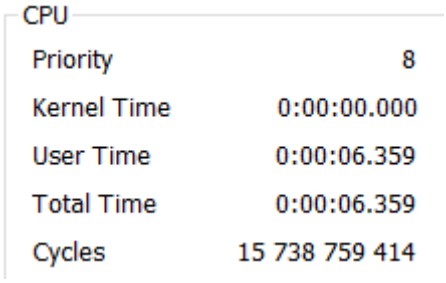
```
void random_read_benchmark(const std::string& file_path, int iterations) {
    std::ifstream file(file_path, std::ios::binary);
    if (!file) {
        std::cerr << "Failed to open file: " << file_path << std::endl;
        return;
    }

    // Определяем размер файла
    file.seekg(0, std::ios::end);
    size_t file_size = file.tellg();
    file.seekg(0, std::ios::beg);

    if (file_size < BLOCK_SIZE) {
        std::cerr << "The file is too small for testing" << std::endl;
        return;
    }

    std::vector<char> buffer(BLOCK_SIZE);
    std::vector<size_t> offsets(iterations);

    // Генерация случайных позиций для чтения
    for (int i = 0; i < iterations; ++i) {
        offsets[i] = (std::rand() % (file_size / BLOCK_SIZE)) * BLOCK_SIZE;
    }

    // Измерение задержек чтения
    auto start_time = std::chrono::high_resolution_clock::now();
    for (int i = 0; i < iterations; ++i) {
        file.seekg(offsets[i]);
        file.read(buffer.data(), BLOCK_SIZE);
        if (!file) {
            std::cerr << "Error read file." << std::endl;
```

```
            break;
        }
    }
    auto end_time = std::chrono::high_resolution_clock::now();

    std::chrono::duration<double> duration = end_time - start_time;
    std::cout << "Average read latency: " << (duration.count() / iterations) *
1e6 << " mks" << std::endl;
}
-------------------------------------------------------------------------------
void factorize_number(unsigned long long number, int iterations) {
    for (int i = 0; i < iterations; ++i) {
        unsigned long long n = number;
        std::vector<unsigned long long> factors;

        for (unsigned long long d = 2; d * d <= n; ++d) {
            while (n % d == 0) {
                factors.push_back(d);
                n /= d;
            }
        }
        if (n > 1) {
            factors.push_back(n);
        }
    }
}
```

BenchmarkFactor
Предположения

Частота процессора 2.5 ГГц
Запустим benchFactor с числом 52. Для выполнения понадобится примерно 100 циклов.
Запустим программу с 10 миллионами итераций.
Регистры 1 / 2.5 = 0.4 ns
100 * 10^7 * 0.4 * (1+4) / 10^9 = 2 sec

```
Execution time: 0.742238 seconds
> C:\Users\MSI\Desktop\OS\testsCpp\build\benchmark\benchmarkFactor.exe r 52 10000000
Execution time: 6.3845 seconds
>
```



Было неверно оценено количество циклов. В итоге чуть не верный результата, но порядок тот же.

15.7 * 10^9 * 0.4ns / 10^9 = 6.28sec

Рассмотрим показатели при разном количестве нагрузчиков.
Все ядра эффективно нагружены при количестве процессов = 12.
USER=97, SYSTEM=3. При увеличении количества процессов в 2 3 4 раза показатели практически не меняются. Но если сделать 10 000 процессов то SYS вырастает до 30%

```
> C:\Users\MSI\Desktop\OS\testsCpp\build\benchmark\benchmarkFactor.exe 1 52 1000000
Execution time: 0.637756 seconds
> C:\Users\MSI\Desktop\OS\testsCpp\build\benchmark\benchmarkFactor.exe 6 52 1000000
Execution time: 0.683415 seconds
> C:\Users\MSI\Desktop\OS\testsCpp\build\benchmark\benchmarkFactor.exe 7 52 1000000
Execution time: 0.762487 seconds
> C:\Users\MSI\Desktop\OS\testsCpp\build\benchmark\benchmarkFactor.exe 12 52 1000000
Execution time: 1.15862 seconds
> C:\Users\MSI\Desktop\OS\testsCpp\build\benchmark\benchmarkFactor.exe 24 52 1000000
Execution time: 2.03613 seconds
> C:\Users\MSI\Desktop\OS\testsCpp\build\benchmark\benchmarkFactor.exe 36 52 1000000
Execution time: 2.28865 seconds
> C:\Users\MSI\Desktop\OS\testsCpp\build\benchmark\benchmarkFactor.exe 48 52 1000000
Execution time: 4.04606 seconds
```

Line #	New Process	Count	CPU Usage (...	Ready (s)	Waits (s)	Count:...	% CPU Usage	Lege
1	benchmarkFactor.exe (19004)	17	0,656544000	0,000246200	1,959496100	17	0,12	
2	benchmarkFactor.exe (17208)	18	0,667497300	0,000427900	1,993563700	16	0,12	
3	benchmarkFactor.exe (18720)	19	0,633850700	0,000741200	1,883191400	18	0,12	
4	benchmarkFactor.exe (5380)	19	0,660742300	0,000463300	1,972777700	19	0,12	
5	benchmarkFactor.exe (19124)	20	0,700210000	0,000246300	2,089579000	20	0,13	
6	benchmarkFactor.exe (19424)	20	0,785054900	0,000802000	2,332389300	20	0,14	
7	benchmarkFactor.exe (19432)	21	0,688022900	0,000423100	2,052157100	18	0,13	
8	benchmarkFactor.exe (19184)	21	0,717123900	0,000312400	2,139558900	20	0,13	
9	benchmarkFactor.exe (18560)	22	0,679288500	0,000322300	2,023136300	19	0,12	
10	benchmarkFactor.exe (19212)	22	0,739172000	0,000465400	2,198888400	17	0,14	
11	benchmarkFactor.exe (15492)	25	0,674376800	0,000693300	2,004574500	22	0,12	
12	benchmarkFactor.exe (18664)	29	0,666418400	0,000406500	1,985921500	20	0,12	
13	benchmarkFactor.exe (18604)	30	0,849567600	0,000413500	2,529687700	21	0,16	
14	benchmarkFactor.exe (19208)	30	1,103502500	0,861341700	3,844095000	11	0,20	
15	benchmarkFactor.exe (18520)	31	1,104570700	0,007075800	2,200419000	10	0,20	
16	benchmarkFactor.exe (19784)	43	1,113253100	2,291241000	6,572148900	9	0,20	
17	benchmarkFactor.exe (18776)	45	1,117977700	1,042313100	3,996316000	10	0,20	
18	benchmarkFactor.exe (12104)	54	1,104936000	0,023015000	3,370862300	18	0,20	
19	benchmarkFactor.exe (19292)	57	1,112330400	0,859653400	3,880362800	10	0,20	
20	benchmarkFactor.exe (17452)	68	1,110520600	0,044295300	2,281777200	17	0,20	
21	benchmarkFactor.exe (19700)	70	1,109297500	1,878958600	5,886068000	9	0,20	
22	benchmarkFactor.exe (15296)	76	1,103606000	0,009667600	3,310922200	21	0,20	

Количество переключений контекста растет пропорционально с увеличением числа нагрузчиков. Но количество переключений в секунду при 100 и при 10 000 процессов одинаковое.

Запустим код с агрессивной оптимизацией –O3 -march=native -ffast-math –flto

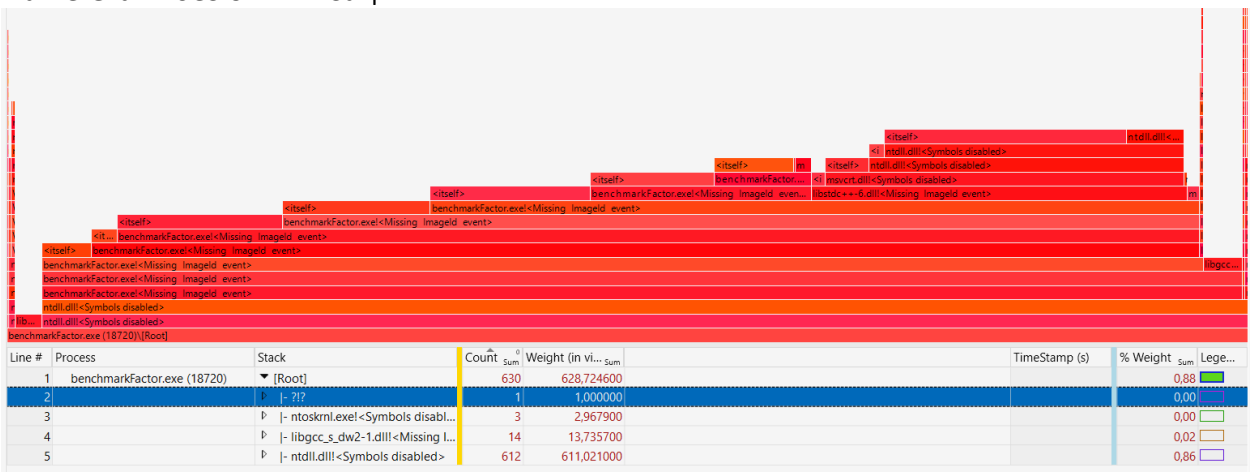
```
> C:\Users\MSI\Desktop\OS\testsCpp\build\benchmark\benchmarkFactor.exe 1 52 10000000
Execution time: 2.29314 seconds
```

Код начал выполняться в 3 раза быстрее благодаря использованию всех доступных инструкции для текущей архитектуры и быстрых математических вычислений. Также видно что пропорционально упало количество циклов для исполнения программы.

Cycles	4 319 788 564

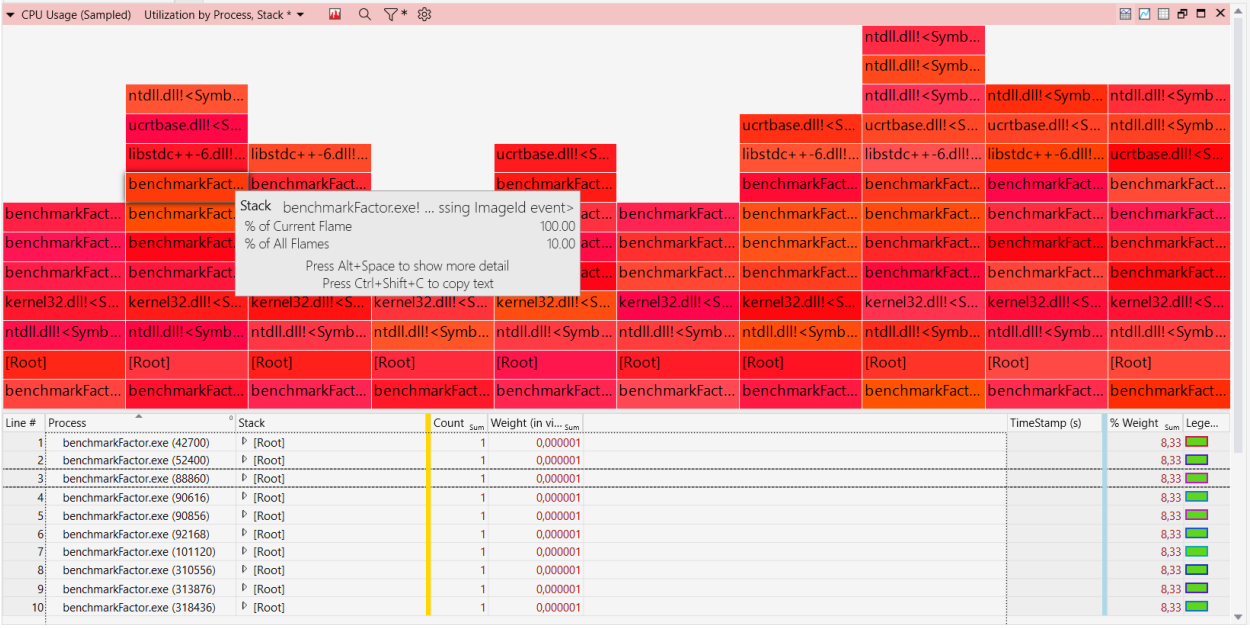
При этом оптимизации на взаимодействие нескольких процессов не произошло.

Flame Graf – без оптимизаций



Line #	Process	Stack	Count Sum	Weight (in vi... Sum	TimeStamp (s)	% Weight Sum	Lege...
1	benchmarkFactor.exe (18720)	[Root]	630	628,724600		0,88	
2		|- ?!?	1	1,000000		0,00	
3		|- ntoskrnl.exe!<Symbols disabl...	3	2,967900		0,00	
4		|- libgcc_s_dw2-1.dll!<Missing I...	14	13,735700		0,02	
5		|- ntdll.dll!<Symbols disabled>	612	611,021000		0,86	

Flame Graf с оптимизациями



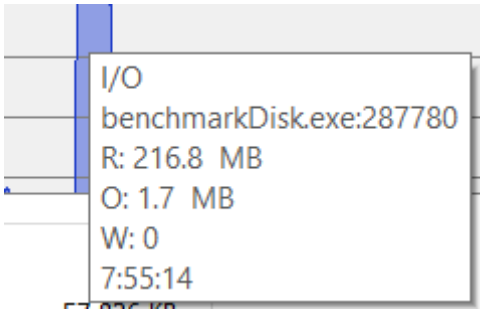
Line #	Process	Stack	Count Sum	Weight (in vi... Sum	TimeStamp (s)	% Weight Sum	Lege...
1	benchmarkFactor.exe (42700)	[Root]	1	0,000001		8,33	
2	benchmarkFactor.exe (52400)	[Root]	1	0,000001		8,33	
3	benchmarkFactor.exe (88860)	[Root]	1	0,000001		8,33	
4	benchmarkFactor.exe (90616)	[Root]	1	0,000001		8,33	
5	benchmarkFactor.exe (90856)	[Root]	1	0,000001		8,33	
6	benchmarkFactor.exe (92168)	[Root]	1	0,000001		8,33	
7	benchmarkFactor.exe (101120)	[Root]	1	0,000001		8,33	
8	benchmarkFactor.exe (310556)	[Root]	1	0,000001		8,33	
9	benchmarkFactor.exe (313876)	[Root]	1	0,000001		8,33	
10	benchmarkFactor.exe (318436)	[Root]	1	0,000001		8,33	

Сразу видна разница весь стек вызова для каждого процесса заполнен во всю ширину на 100%

BenchmarkDisk
Предположения

Скорость моего SSD = 2200 МБ/с
1/(2200 * 10^6) * 1024 = 0,46mks

```
> C:\Users\MSI\Desktop\OS\testsCpp\build\benchmark\benchmarkDisk.exe 1 C:\Users\MSI\Desktop\PremierePro\Adobe.Premiere.Pro.2024.u3.Multilingual.iso 200000
Average read latency: 4.755 mks
Execution time: 0.966007 seconds
```



Видим, что скорость была в 10 раз меньше. Результат соответствующий.
Далее умножаем на 200 000 и получаем финальное время исполнения.

Задержка на чтение увеличивается пропорционально количеству процессов.

Процессор практически не задействован в данном нагрузчике поэтому чтобы загрузить его на 100 процентов нужно запустить 64 процесса.
64:

```
Average read latency: 169.331 mks
Execution time: 3.6314 seconds
```

128

```
Execution time: 0.848039 seconds
```
Average read = 150-256

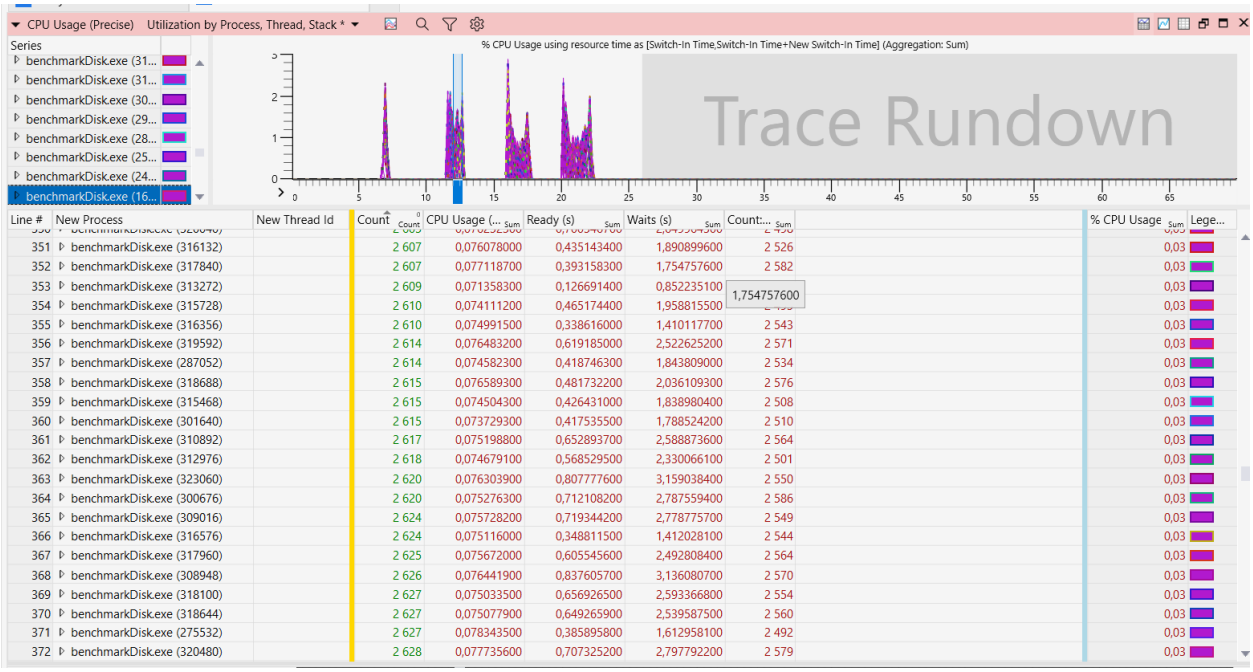
192

```
Execution time: 1.23686 seconds
```
Average read = 150-310

256

```
Execution time: 1.67221 seconds
```
Average read = 150-314

Ситуация обратная banchFactor. USER=3% SYS=97%

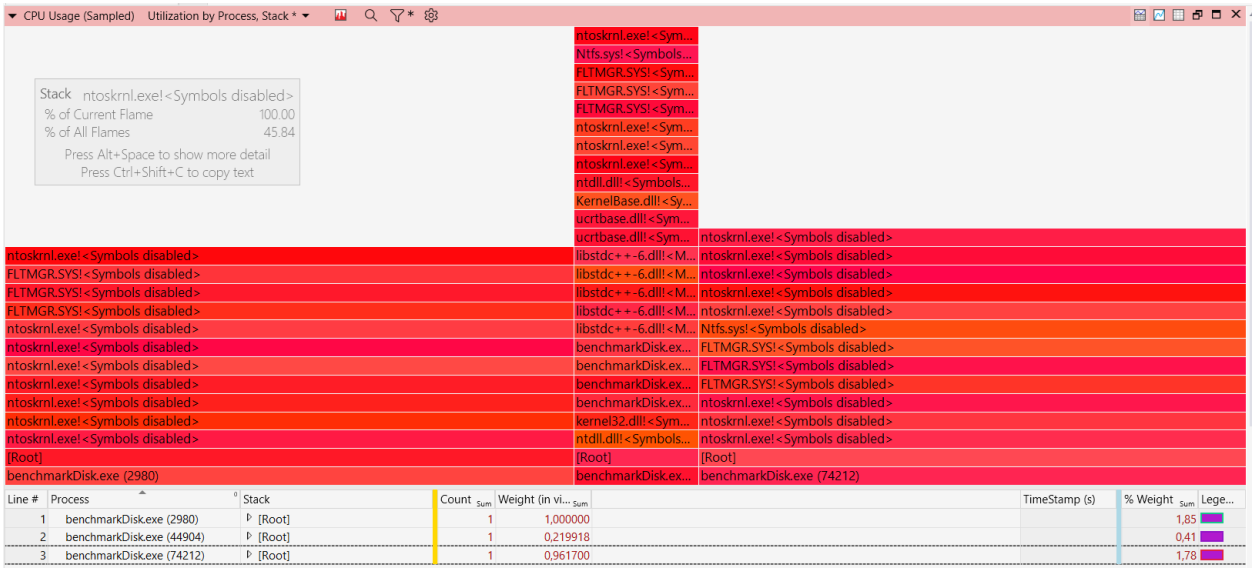


Количество переключений контекста порядок одинаковый (для процесса) что для 64 что для 256

Flame Graf

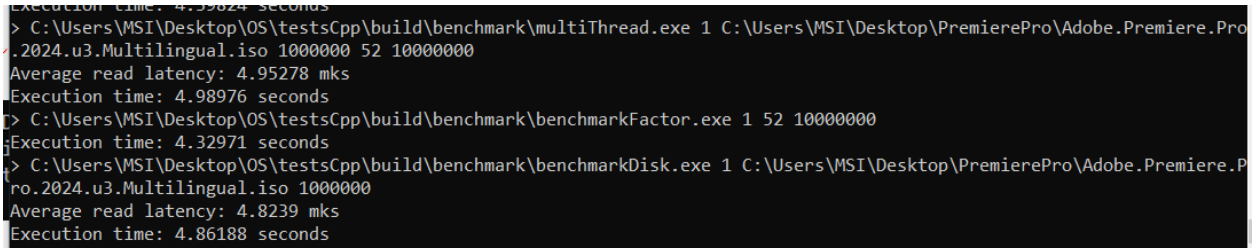
BenchmarkDisk с оптимизациями –O3 -march=native -ffast-math –funroll-loops
Ничего не изменило т.к. в основном идет нагрузка на I/O а не на CPU.

Flame Graf с оптимизациями

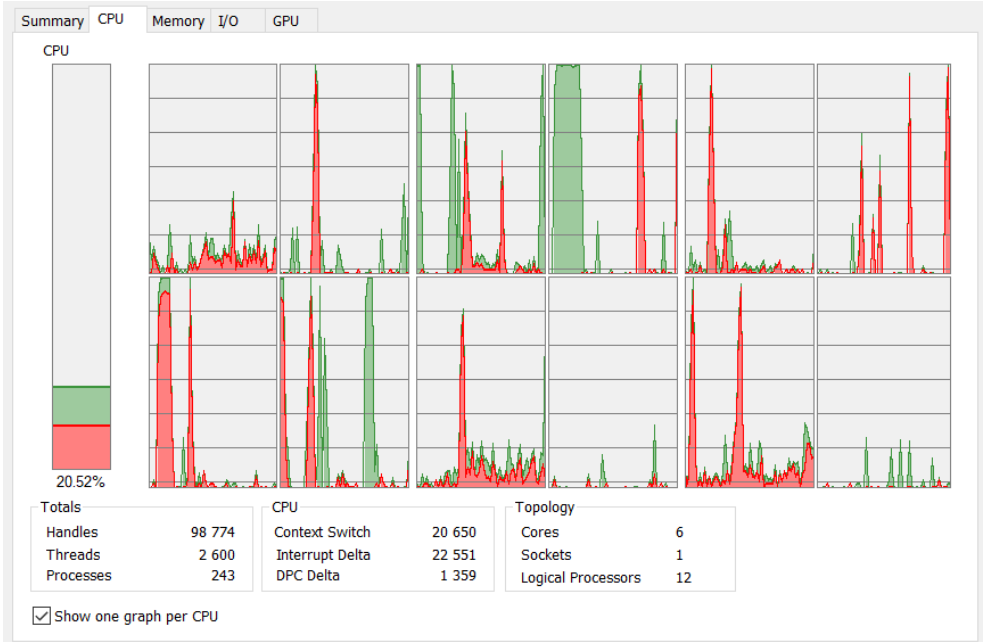


Разницы практически нет кроме уменьшения глубины стека но на результате это не отображается.

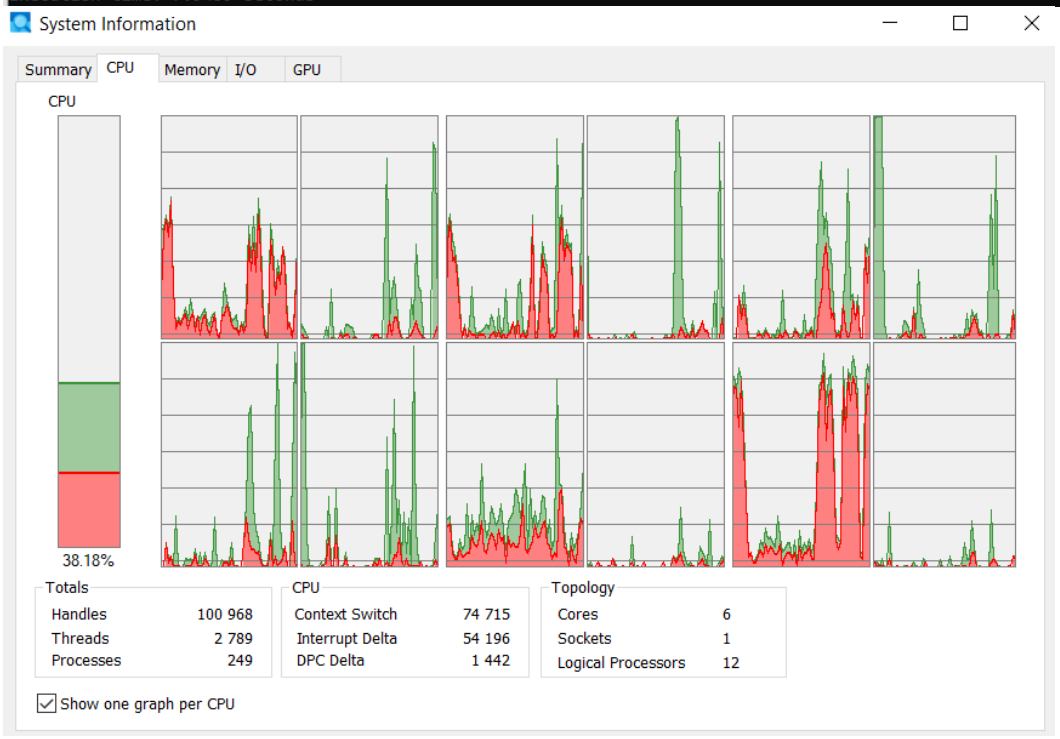
MultiThread



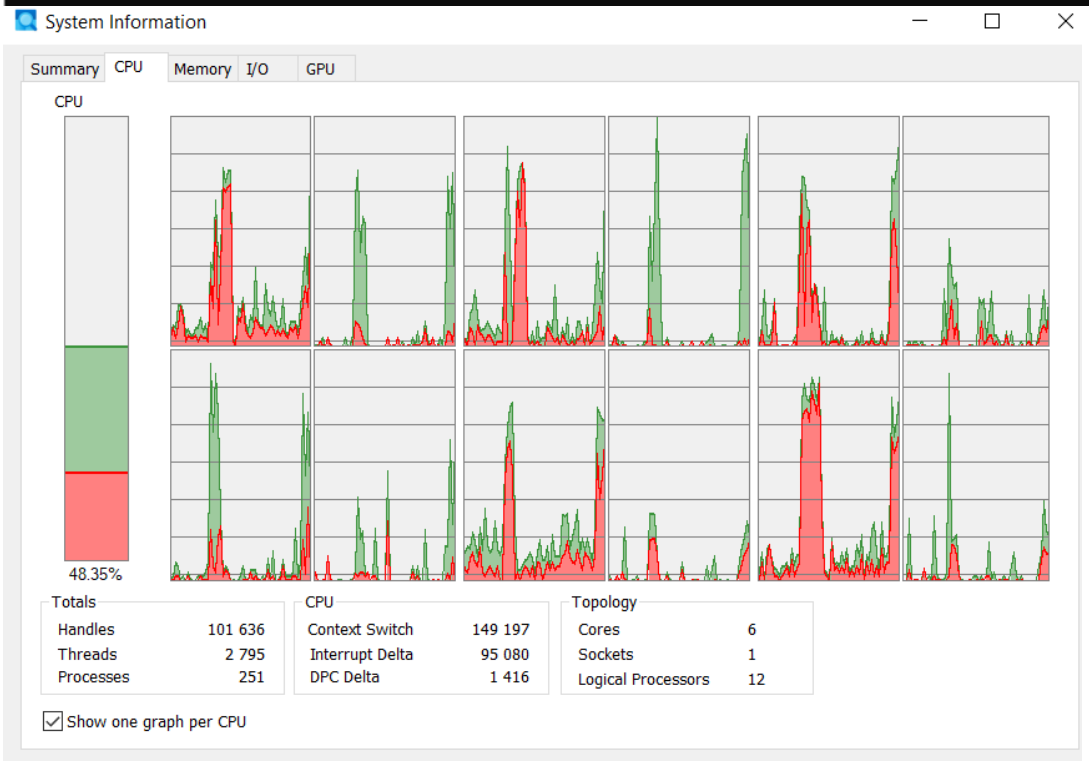
При запуске двух нагрузчиках при том же объеме данных скорость осталось той же за счет параллельности. Загрузка процессора при этом 20% при увеличении количества итераций увеличивается пропорционально время исполнения (достигнута максимальная параллельность) поэтому увеличиваем число таких программ.



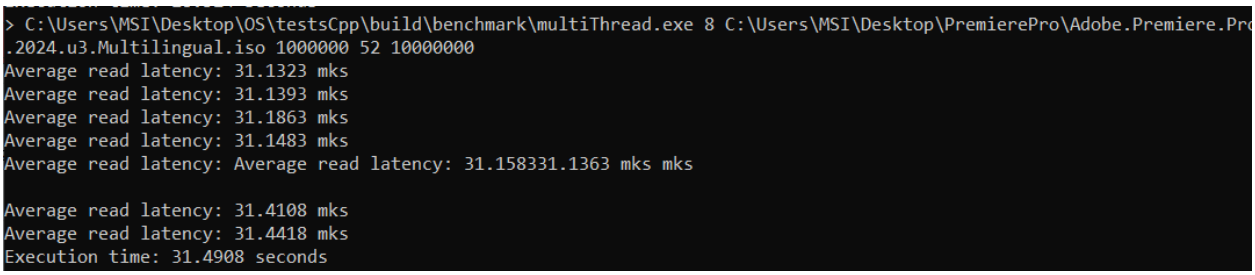
```
> C:\Users\MSI\Desktop\OS\testsCpp\build\benchmark\multiThread.exe 2 C:\Users\MSI\Desktop\PremierePro\Adobe.Premiere.Pro
.2024.u3.Multilingual.iso 1000000 52 10000000
Average read latency: 7.90593 mks
Average read latency: 7.88993 mks
Execution time: 7.9439 seconds
```




```
> C:\Users\MSI\Desktop\OS\testsCpp\build\benchmark\multiThread.exe 3 C:\Users\MSI\Desktop\PremierePro\Adobe.Premiere.Pro
.2024.u3.Multilingual.iso 1000000 52 10000000
Average read latency: 9.6538 mks
Average read latency: 9.6578 mks
Average read latency: 9.97541 mks
Execution time: 10.0194 seconds
```



Нагрузим процессор максимально для этого нужно 8 программ



Дополнительно всю информация по нагрузчикам можно посмотреть в WPA для этого было сформировано 3 дампа при помощи WPR.
Весят они 3ГБ демонстрация только офлайн)