Лабораторная работа №1

Выполнил: Трошкин Александр Евгеньевич, 333304
Группа: P3316
Преподаватель: Гиниятуллин Арслан Рафаилович
Вариант:
`Windows`, `CreateProcessAsUser`, `ema-search-int`, `short-path`
Желаемые баллы: минимум

Задание

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?

Ограничения

Программа (комплекс программ) должна быть реализован на языке C, C++. Дочерние процессы должны быть созданы через заданные системные вызовы выбранной операционной системы, с обеспечением корректного запуска и и завершения процессов. Запрещено использовать высокоуровневые абстракции над системными вызовами. Необходимо использовать, в случае Unix, процедуры libc.

Требования к отчету и защите

- Отчет должен содержать титульный лист с указанием номера и названия ЛР, вашего ФИО, ФИО преподавателя практики, номера вашей группы, варианта ЛР.
- Отчет должен содержать текст задания в соответствии с вариантом.
- Отчет должен содержать листинг исходного кода всех программ, написанных в рамках данной ЛР.

- Отчет должен содержать предположения о свойствах программ-нагрузчиков
- Отчет должен содержать результаты измерений и метрик программ-нагрузчиков, полученных инструментами мониторинга. Должно быть описано, какие утилиты запускались, с какими параметрами и выводом.
- Отчет должен содержать сравнительный анализ ожидаемых и фактических значений.
- Отчет должен содержать вывод.
- Студент должен быть готов продемонстрировать работоспособность Shell и предоставить исходный код.
- Студент должен быть готов воспроизвести ход работы в рамках части 2 и продемонстрировать схожие результаты работы программ-нагрузчиков.

Темы для подготовки к защите лабораторной работы

1. Структура процесса и потоков;
2. Системные утилиты сбора статистики ядра;
3. Основы ввода-вывода (блочный и последовательный ввод-вывод);
4. Файловая система procfs;
5. Использование утилиты strace, ltrace, bpftrace;
6. (*) Профилирование и построение flamegraph'a и stap;

Листинг программы

Исходный код можно посмотреть на github: https://github.com/k1nd-cat/os-labs

Предположения о свойствах программ-нагрузчиков

`ema-search-int` -- данный алгоритм ищет случайно расположенный элемент типа int в файле размером 256 Мб.

Возьмём максимальную по нагрузке ситуацию, когда у нас элемент находится в самом конце. В таком случае необходимо будет перебрать примерно 2^25 элементов.

Учитывая, что алгоритм поиска элемента выполняется примерно за ~5asm инструкций, а мой процессор обладает частотой 2.8 GHz, то получаем примерно 560 миллионов элементов / сек.

Таким образом 2^25 / 560 M = 60 мс.

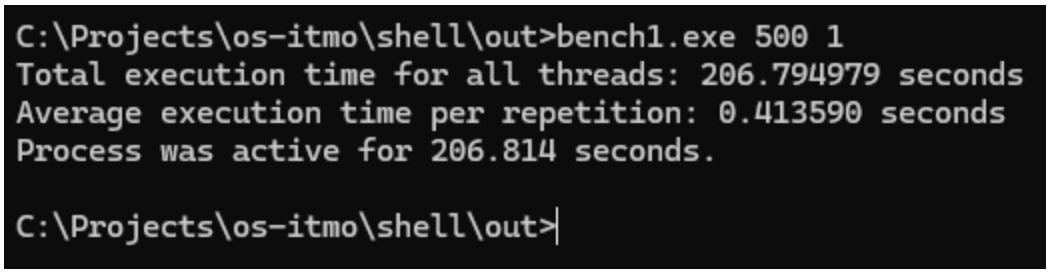
Поскольку чтение файла происходит блоками по 32 Мб, а скорость SSD примерно равна 3000 Мб/сек, то 256 / 3000 = 85 мс уйдёт на чтение файла. Таким образом среднее время выполнения 85 + 60 = 145 мс.

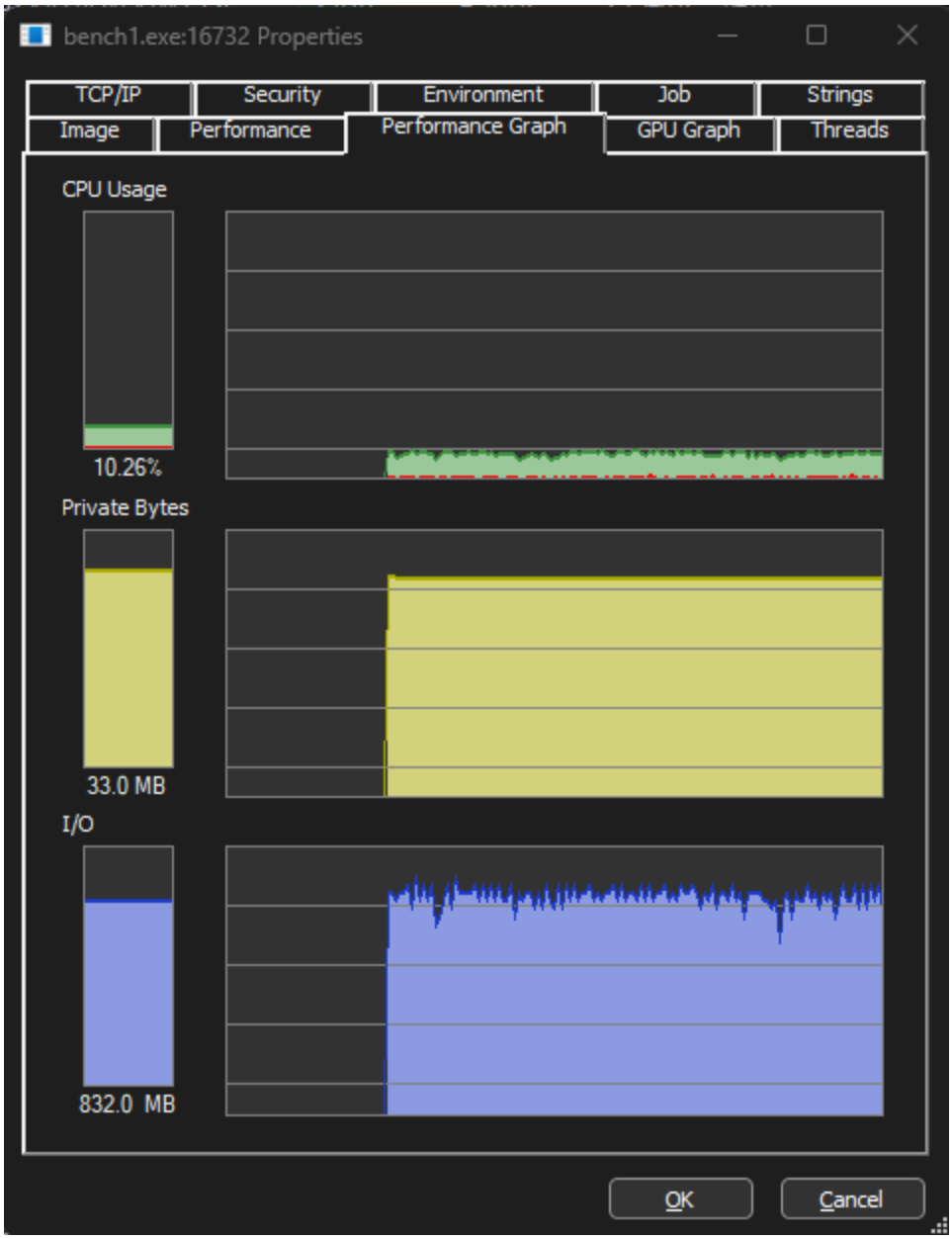
`short-path` -- у меня данный тест реализует алгоритм Дейкстры для поиска кратчайшего пути от одной вершины ко всем остальным.

В среднем на одну итерацию выходит примерно 100 инструкций. Если сложность алгоритма O((V + E) * log(V)), 10000 вершин и примерно 24 997 500 рёбер, то для поиска кратчайших путей выйдет 3.3217*10^8 операций или 3.3217*10^10 инструкций. Тогда примерное время выполнения такого алгоритма должно быть равно 1100 мс.

Результаты измерений и метрик программ-нагрузчиков

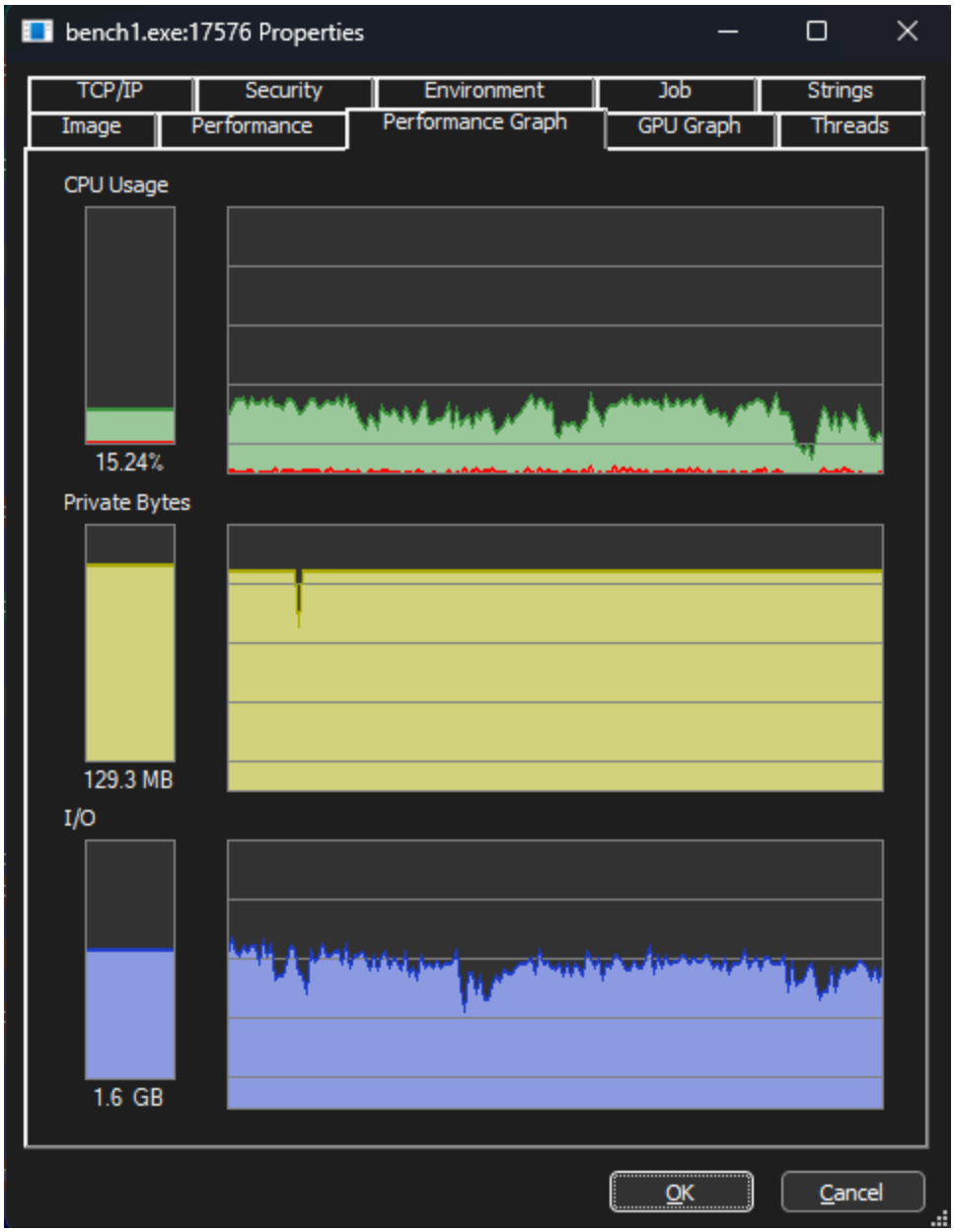
ema-search-int





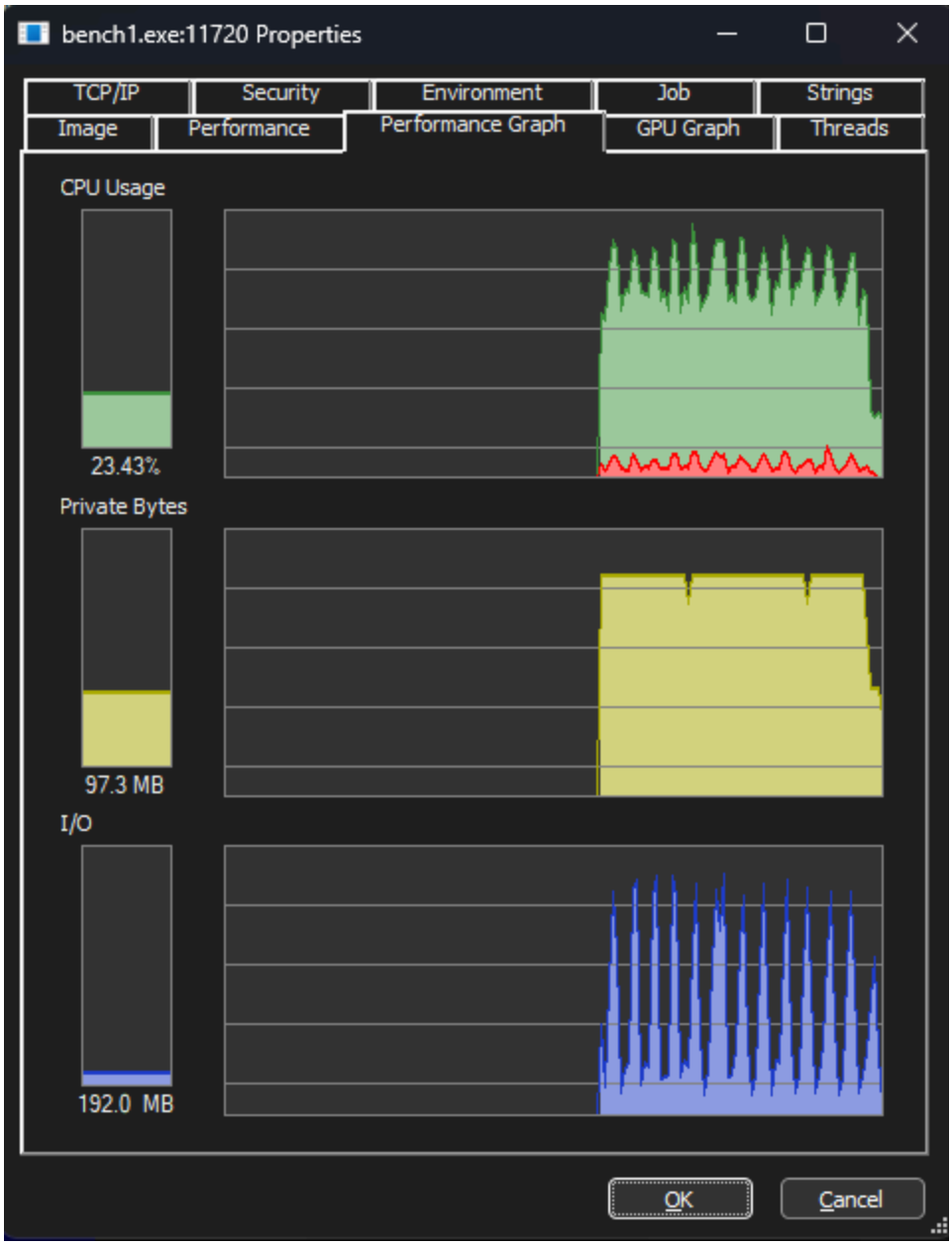
Теперь запустим первый бенчмарк, разделив задачу на 4 потока

```
C:\Projects\os-itmo\shell\out>bench1.exe 500 4
Total execution time for all threads: 1921.709495 seconds
Average execution time per repetition: 0.960855 seconds
Process was active for 481.686 seconds.

C:\Projects\os-itmo\shell\out>
```



Запустим первый бенчмарк, заняв все 8 ядер процессора

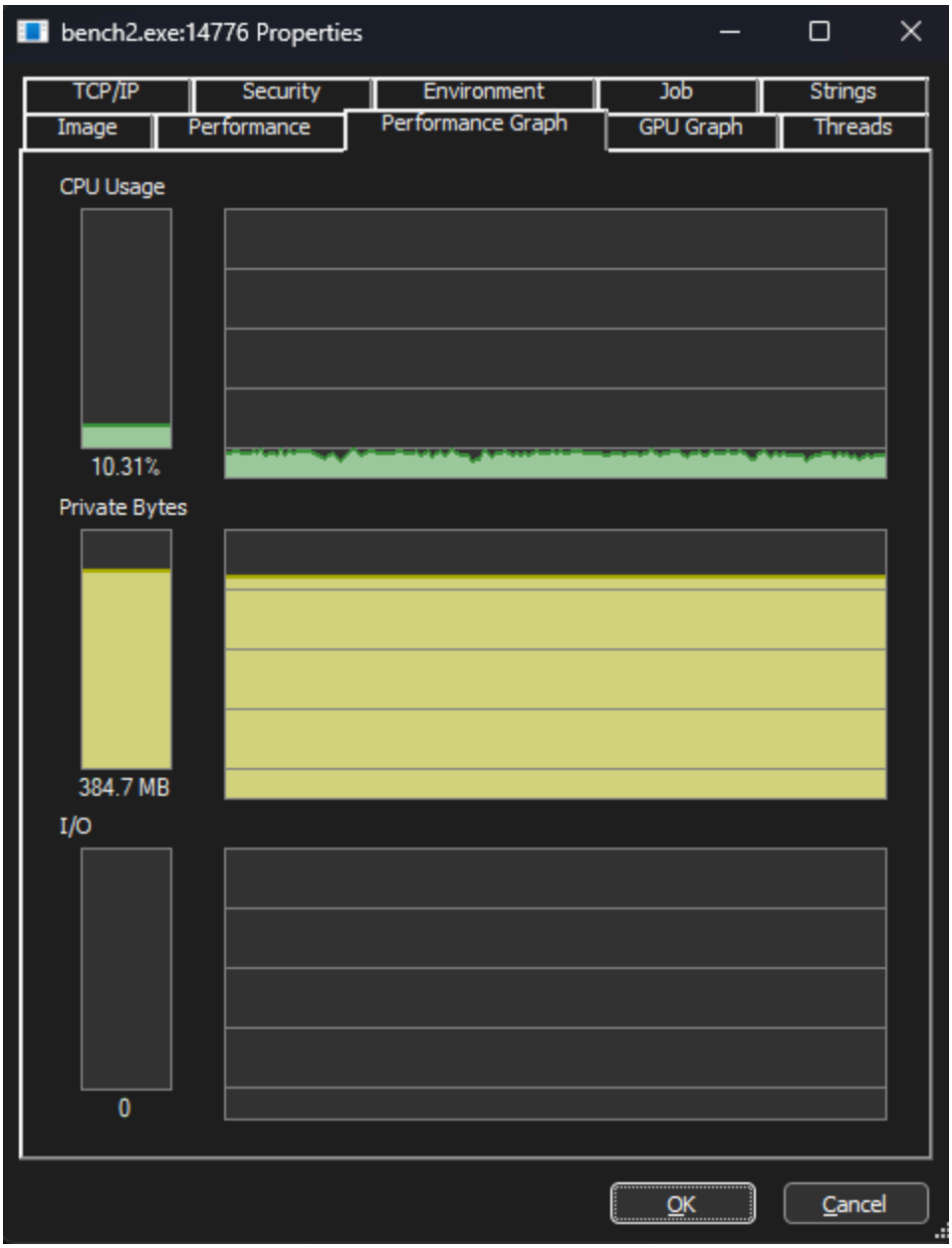
```
C:\Projects\os-itmo\shell\out>bench1.exe 50 8
Total execution time for all threads: 866.457500 seconds
Average execution time per repetition: 2.166144 seconds
Process was active for 111.589 seconds.

C:\Projects\os-itmo\shell\out>
```



short-path

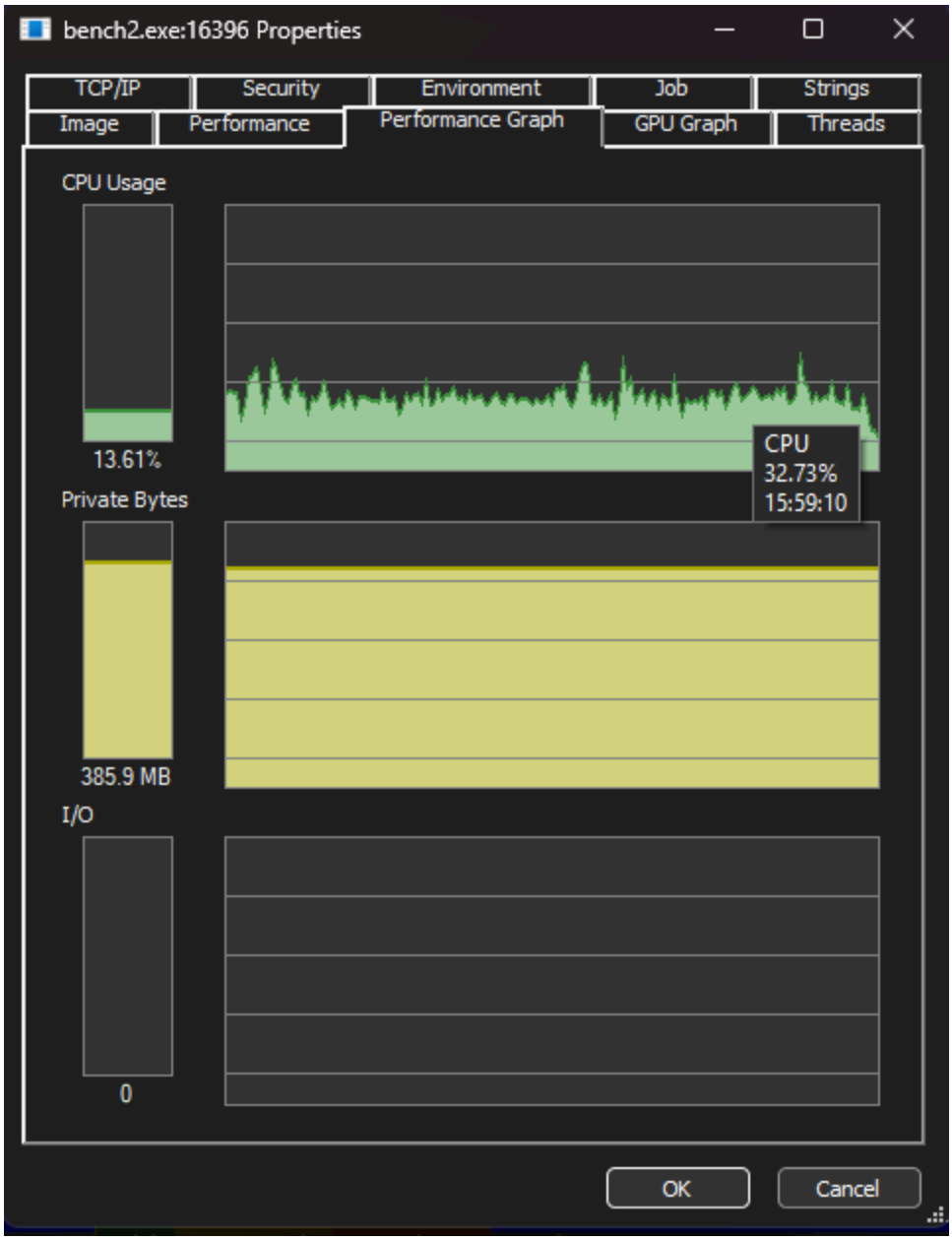
Запуск в 1 потоке

```
C:\Projects\os-itmo\shell\out>bench2.exe 2 500 1
Total execution time for all threads: 593.630007 seconds
Average execution time per repetition: 1.187260 seconds
Process was active for 595.754 seconds.

C:\Projects\os-itmo\shell\out>
```



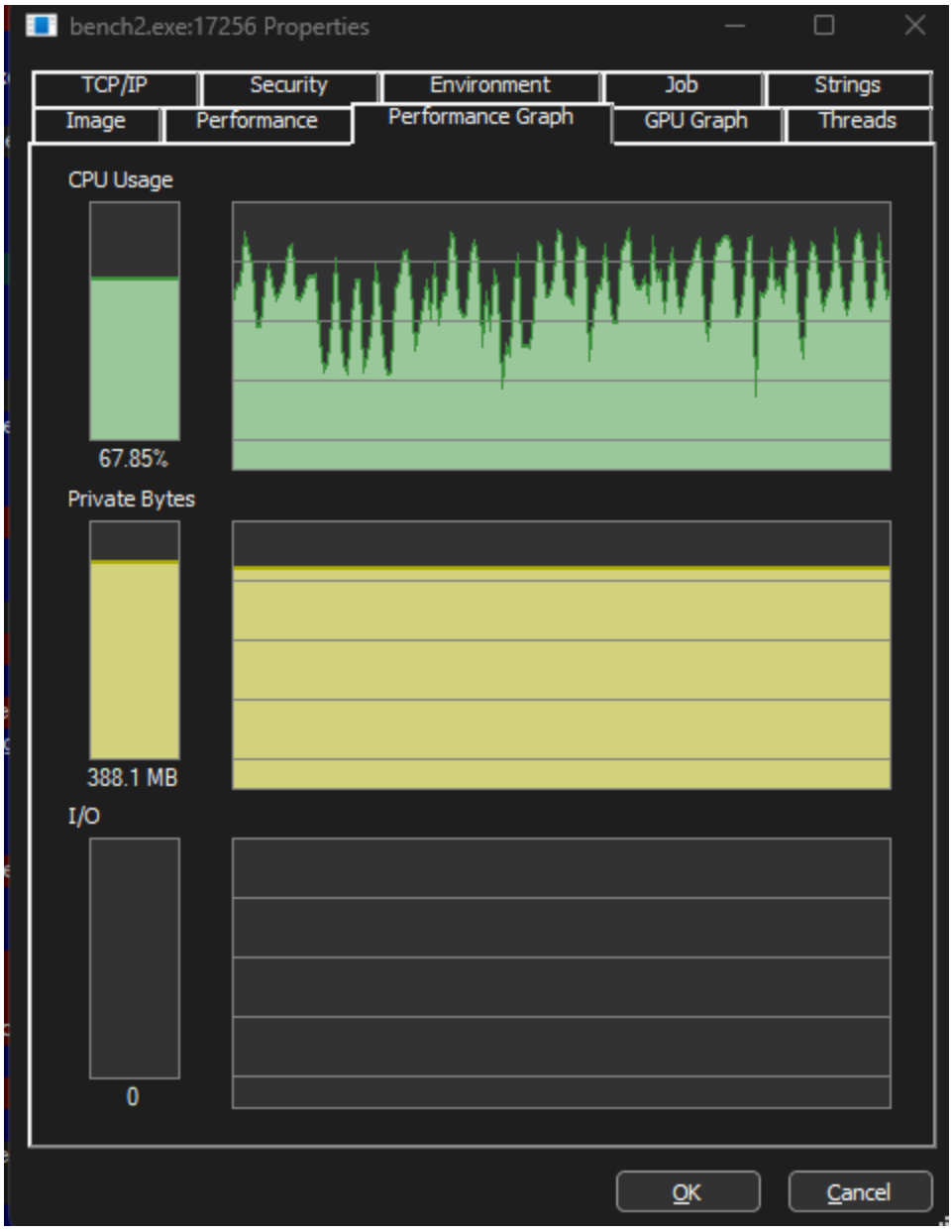
Запуск в 4 потоках

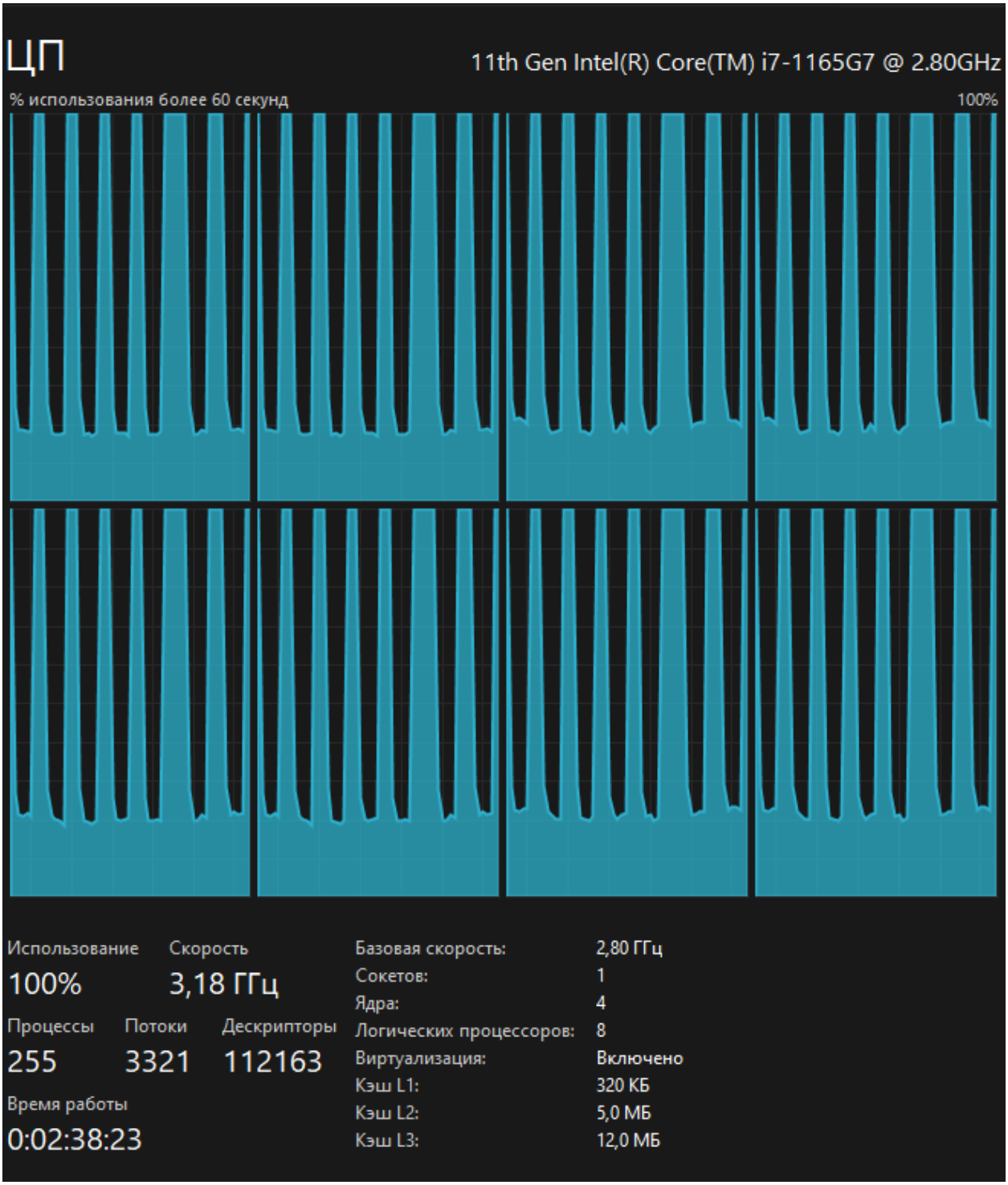
```
C:\Projects\os-itmo\shell\out>bench2.exe 2 100 4
Total execution time for all threads: 1359.221736 seconds
Average execution time per repetition: 3.398054 seconds
Process was active for 347.593 seconds.
```



Запуск в 8 потоках

```
C:\Projects\os-itmo\shell\out>bench2.exe 2 100 8
Total execution time for all threads: 4593.987164 seconds
Average execution time per repetition: 5.742484 seconds
Process was active for 616.353 seconds.

C:\Projects\os-itmo\shell\out>
```

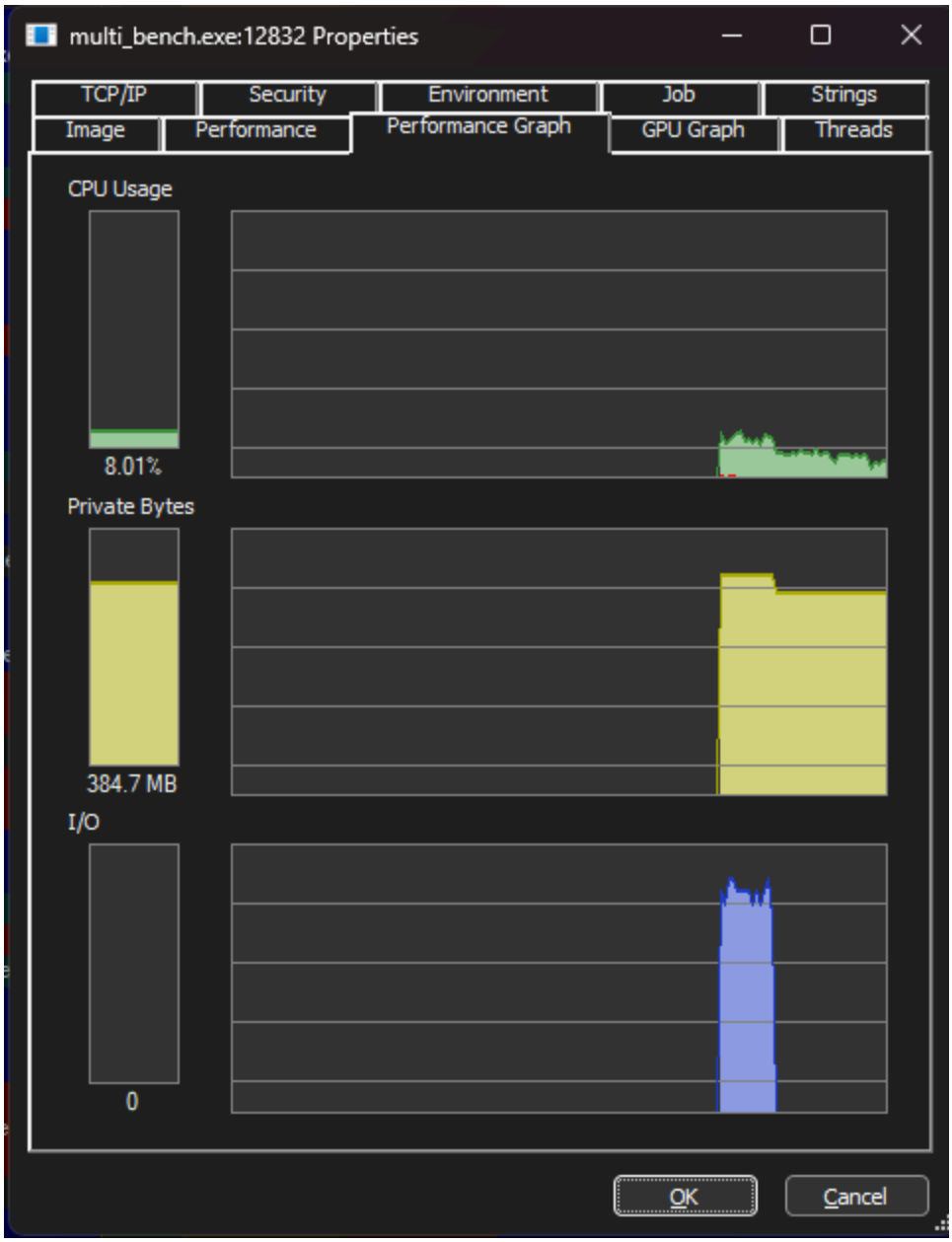

bench2.exe:17256 Properties
TCP/IP
Security
Environment
Job
Strings
Image
Performance
Performance Graph
GPU Graph
Threads
CPU Usage
67.85%
Private Bytes
388.1 MB
I/O
0
OK
Cancel



multi-bench

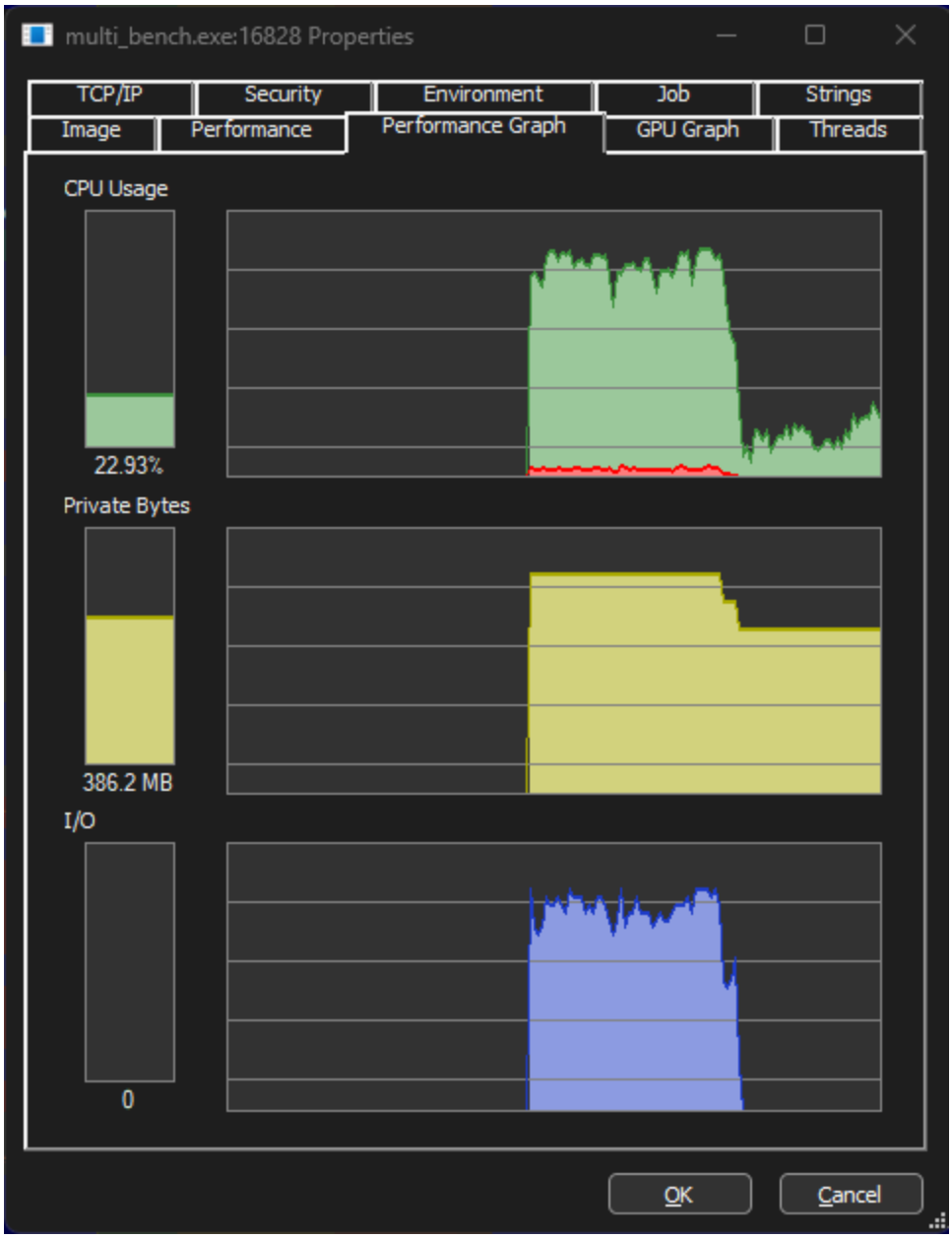
Запуск в 1 потоке

```
C:\Projects\os-itmo\shell\out>multi_bench.exe 50 1
Algorithm: Short Path Search
Total execution time for all threads: 73.745997 seconds
Average execution time per repetition: 1.474920 seconds

Algorithm: Number Search
Total execution time for all threads: 31.347023 seconds
Average execution time per repetition: 0.626940 seconds
Process was active for 77.0016 seconds.
```



Запуск в 4 потоках

```
C:\Projects\os-itmo\shell\out>multi_bench.exe 50 4
Algorithm: Short Path Search
Total execution time for all threads: 676.567379 seconds
Average execution time per repetition: 3.382837 seconds

Algorithm: Number Search
Total execution time for all threads: 355.840804 seconds
Average execution time per repetition: 1.779204 seconds
Process was active for 172.419 seconds.

C:\Projects\os-itmo\shell\out>
```



Вывод

Выполнив лабораторную работу, я научился создавать дочерние процессы на Windows. Написал свою командную оболочку и реализовал в ней встроенные команды: `cd`, `dir`, Запуск сторонних процессов. Также я написал программы-нагрузчики, с помощью которых мониторил и профилировал программы.