Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Операционные системы

Студент: Немкович А.В.

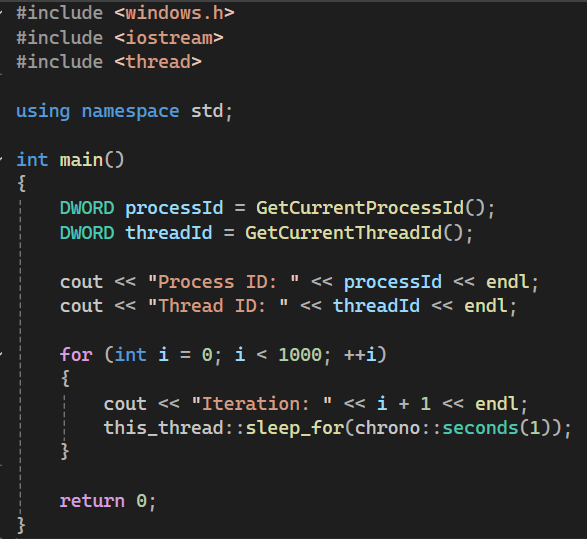
ФИТ 3 курс 1 группа

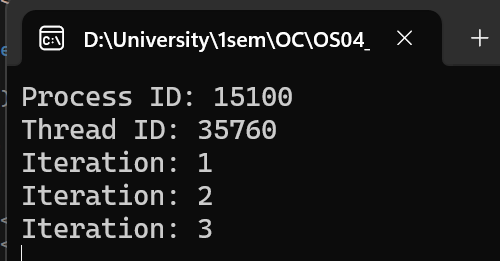
Преподаватель: Бернацкий П.В

Минск 2024

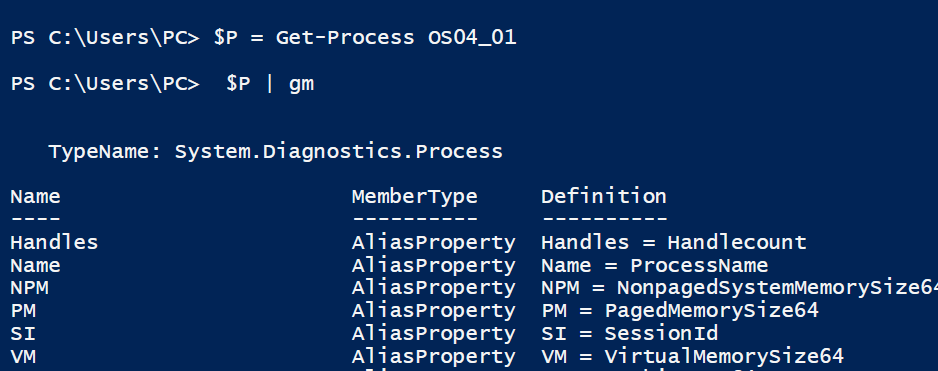
**Задание 01**

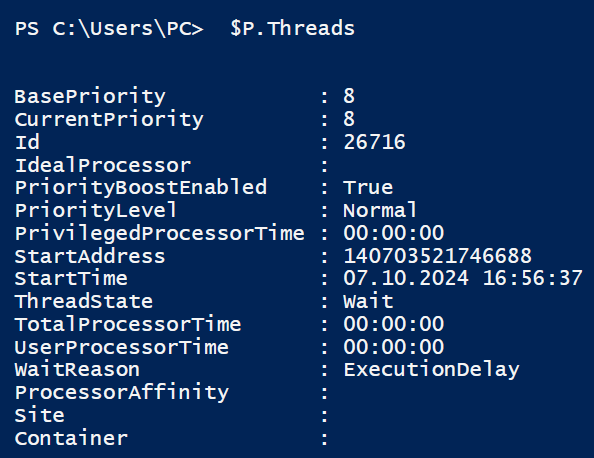
Разработайте консольное Windows-приложение OS04\_01 наязыке С++, выполняющее длинный цикл с временнойзадержкой и с выводом на консоль идентификаторовтекущего процесса и текущего потока.

****

****

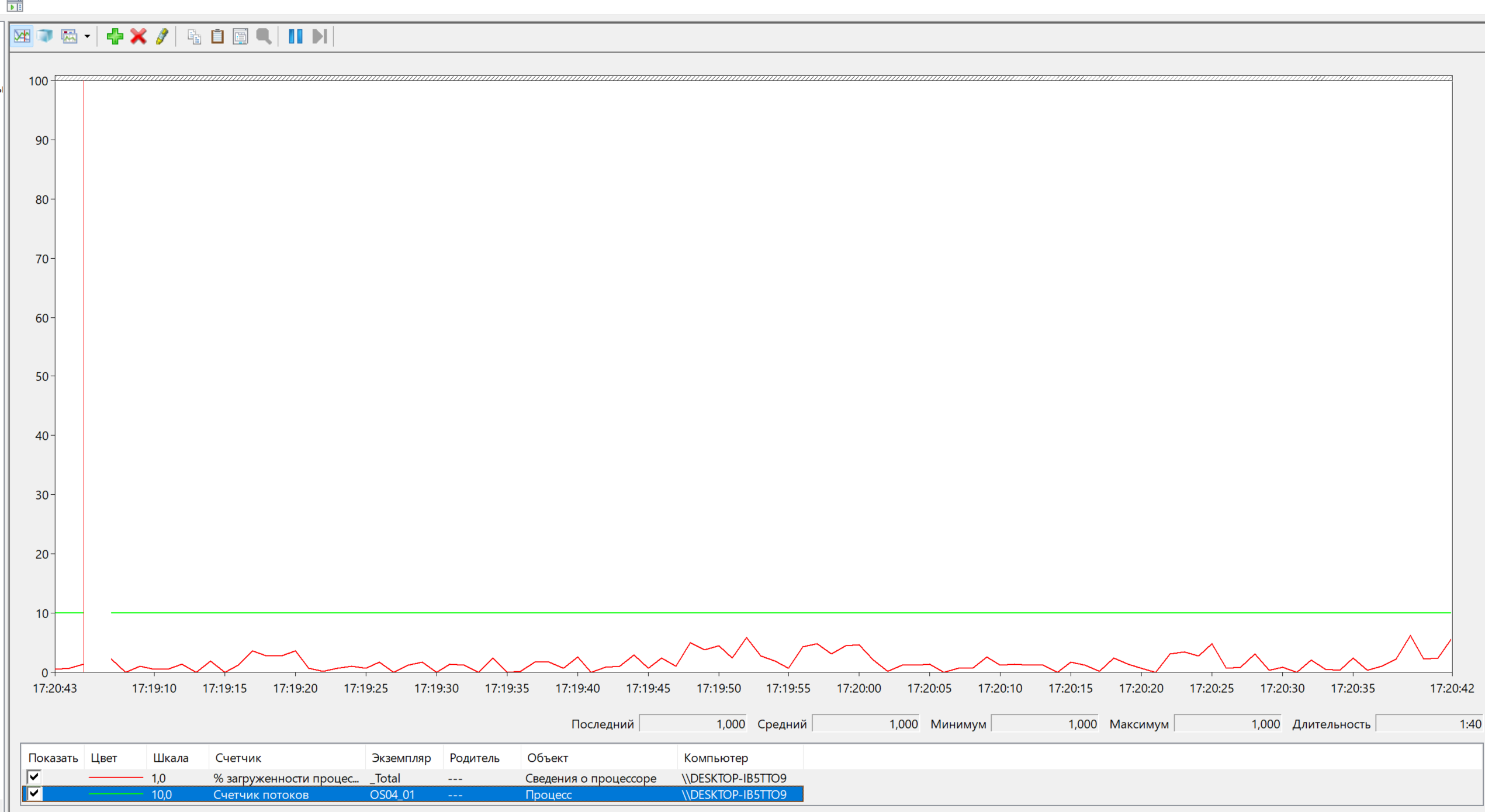
Продемонстрируйте информацию об потоках процесса  
OS04\_01 с помощью утилит PowerShell ISE





* BasePriority: Базовый приоритет потока (8). Это приоритет, с которым поток был создан.
* CurrentPriority: Текущий приоритет потока (8). Он может изменяться в зависимости от системной нагрузки.
* Id: Это уникальный идентификатор потока в системе.
* PriorityBoostEnabled: Флаг, указывающий, включено ли повышение приоритета потока
* PriorityLevel: Уровень приоритета потока. Это приоритетная категория, которая может быть одним из значений: Idle, BelowNormal, Normal, AboveNormal, High, или RealTime.
* PrivilegedProcessorTime: Время процессора, потраченное на выполнение привилегированных операций (системные операции).
* StartAddress: Адрес, с которого начинается выполнение потока.
* StartTime: Время старта потока
* ThreadState: Текущее состояние потока. В данном случае это Wait — поток ожидает ресурсы или выполнение операции. (если каждая итерация цикла включает задержку в 1 секунду (sleep\_for), поток большую часть времени находится в состоянии ожидания, выполняя лишь небольшие вычисления при переходе к следующей итерации. Из-за этого процессорное время, потраченное на выполнение, остается близким к нулю, так как основная активность потока — это ожидание.)
* TotalProcessorTime: Общее время процессора, затраченное потоком на выполнение
* UserProcessorTime: Время, потраченное на выполнение пользовательского кода, как часть общего процессорного времени
* WaitReason: Причина ожидания потока
* ProcessorAffinity: Это свойство может указывать на предпочтительный процессор, на котором должен выполняться поток

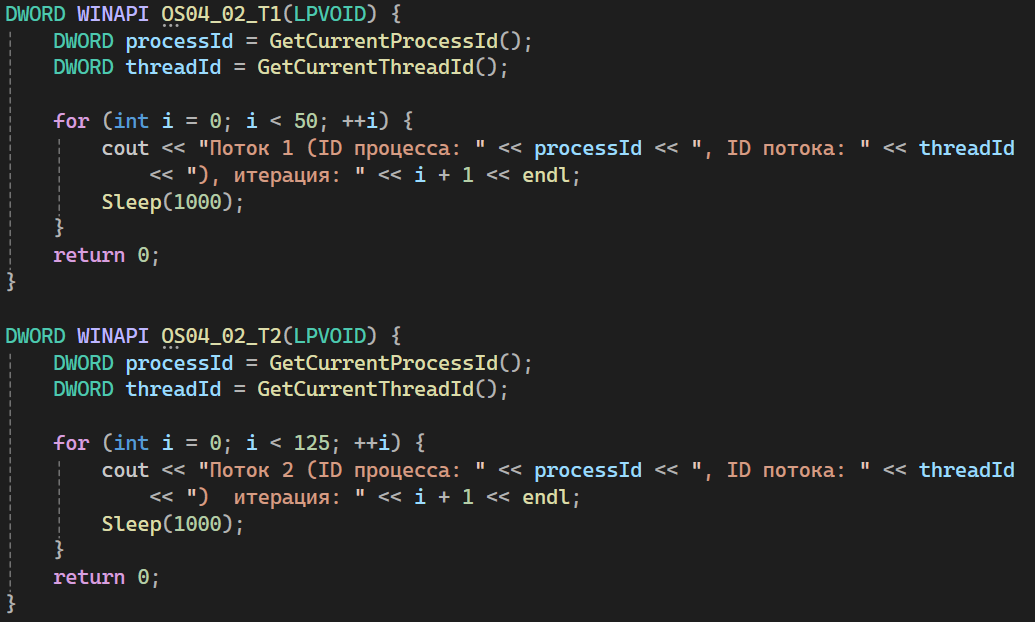
и Performance Monitor (perfmon)

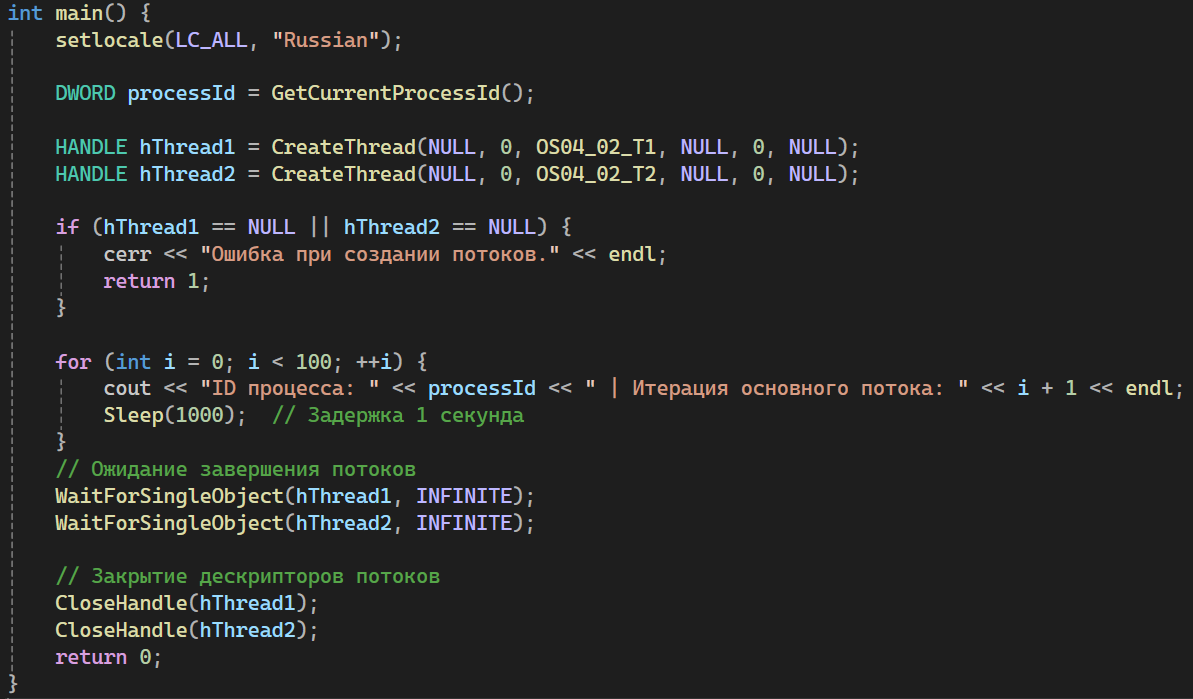


**Задание 02**

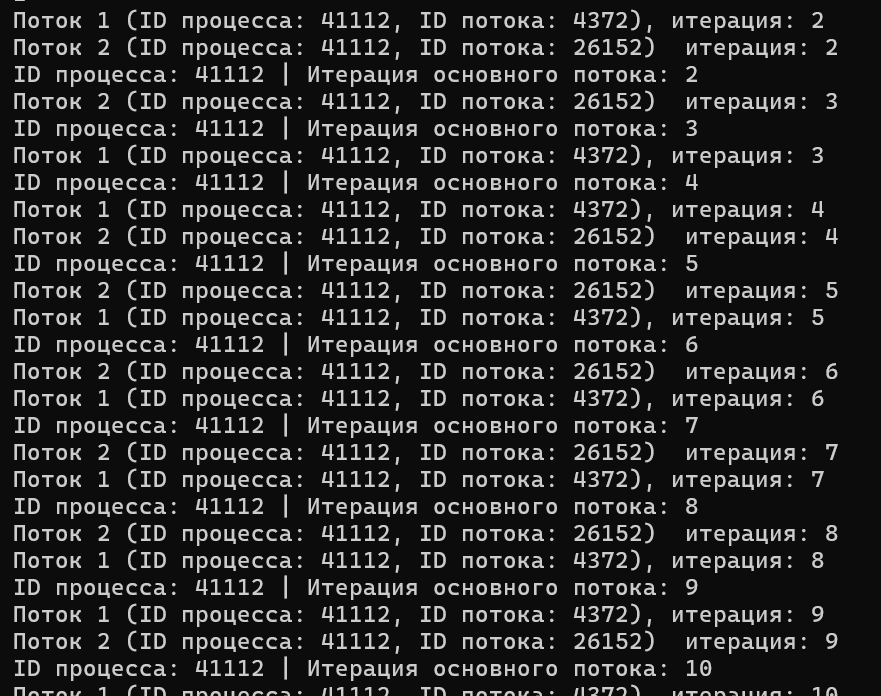
Разработайте консольное Windows-приложение OS04\_02 на языке С++, выполняющее цикл 100 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса.

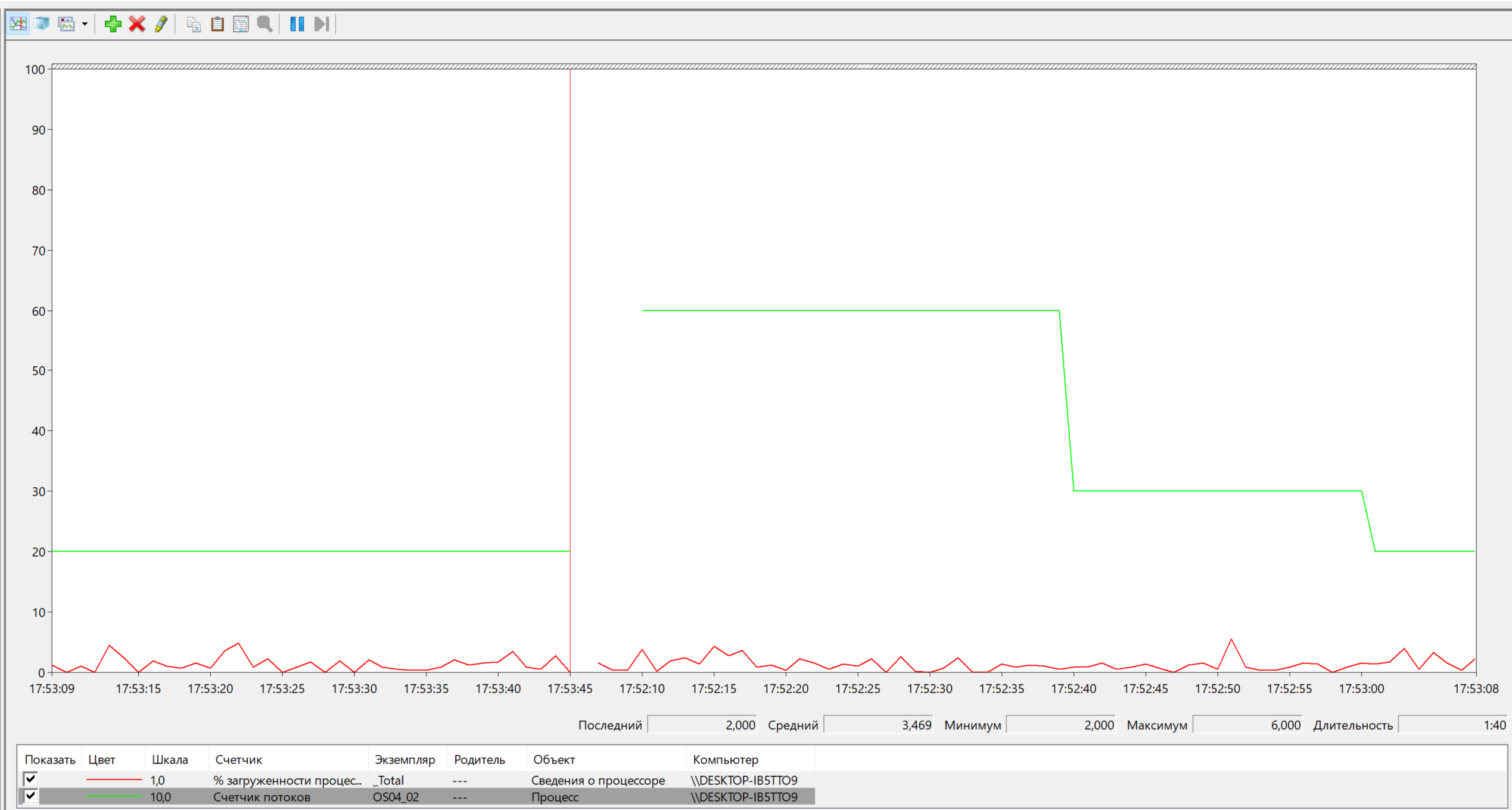
Процесс OS04\_02 должен создать два потока: потоковые функции OS04\_02\_T1, OS04\_02\_T2. Поток OS04\_02\_T1 - выполняет цикл 50 итераций с временной задержкой в 1 сек. с выводом на консоль идентификаторов процесса и потока. Поток OS04\_02\_T2 - выполняет цикл 125 итераций с временной задержкой в 1 сек. с выводом на консоль идентификаторов процесса и потока.

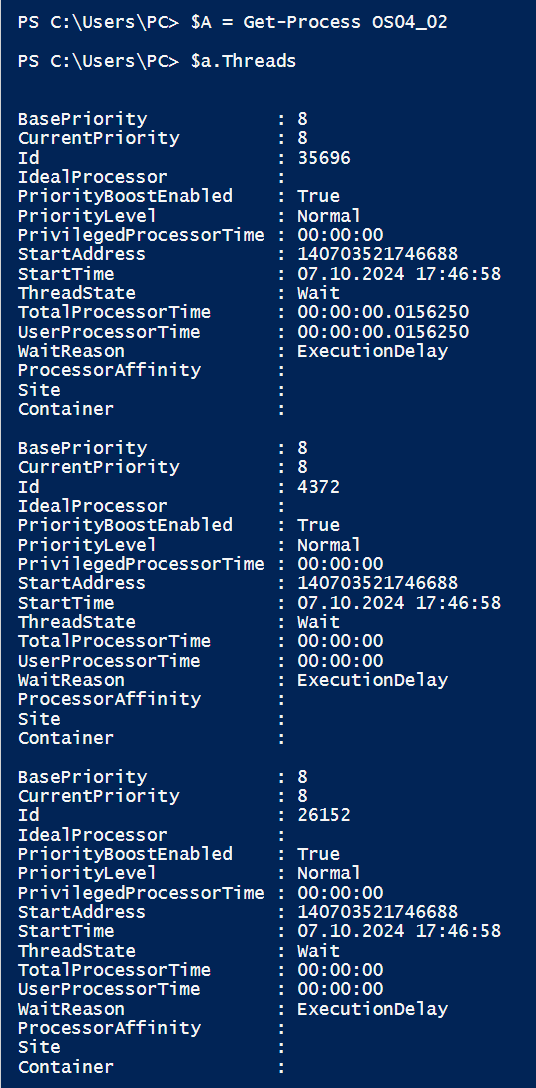




Продемонстрируйте информацию об потоках процесса OS04\_02 с помощью утилит PowerShell ISE и Performance Monitor.

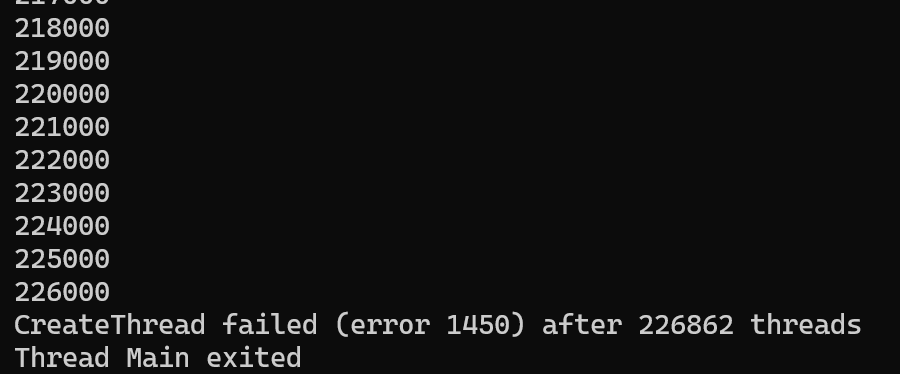






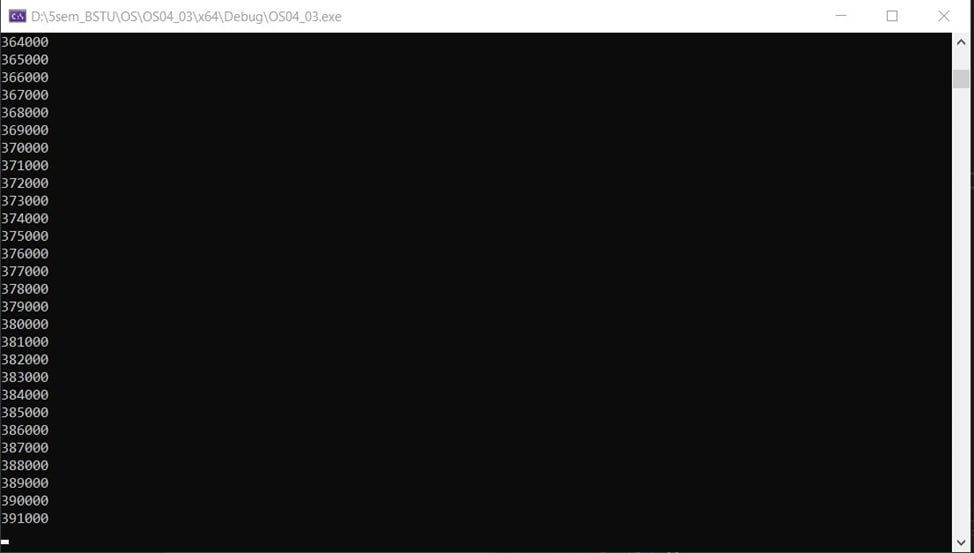
**Задание 03**

Определите, какое максимальное количество потоков можносоздать в одном процессе на вашем компьютере, используянижеприведенный программный код (или аналогичный).Сравните результаты между собой. От чего зависитмаксимальное количество потоков? Какое ограничениенакладывает операционная система.

****

Максимальное количество потоков, которые удалось создать, составило 226862.

Другие результаты:



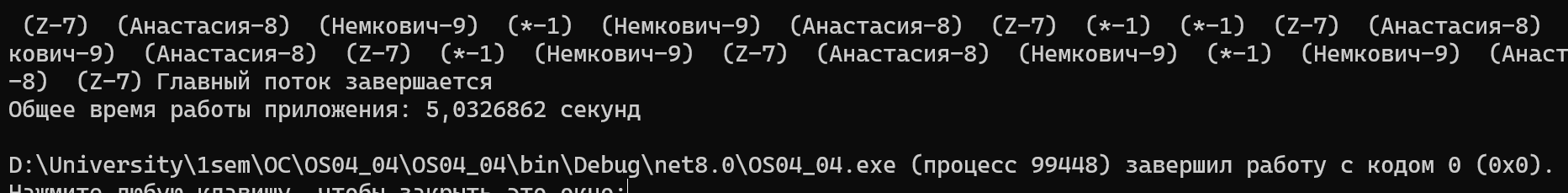
Что может влиять:

* Каждый поток требует определённого объёма памяти для стека. Если система не может выделить необходимое количество памяти, создание новых потоков будет ограничено.
* Количество доступных ядер и процессоров в системе
* Другие запущенные процессы и системные ресурсы

**Задание 04**

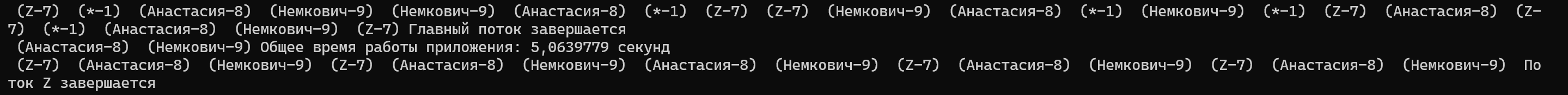
Создайте консольное приложение на C#, которое запускает три дополнительных потока и завершается через пять секунд. Один дополнительный поток завершается через 10 секунд, остальные два – через двадцать секунд. Фрагмент программного кода приведен ниже. (Вставьте Свои Имя-Фамилию).

Выполните приложение. Обратите внимание на общее время работы приложения.



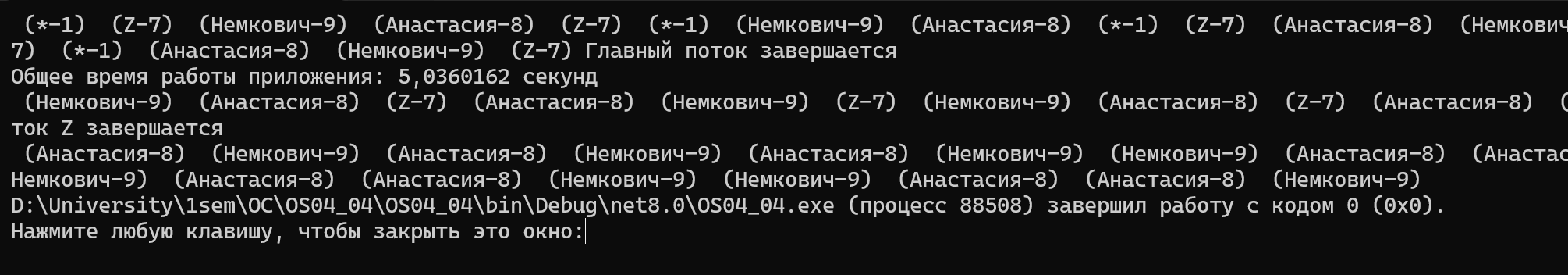
+-5 сек: так как оба дополнительных потока являются фоновыми, они могут быть завершены, когда основной поток завершится.

Измените значение свойства IsBackground для первого дополнительного потока на false и снова выполните приложение. Обратите внимание на общее время работы приложения.



+- 10 сек: общее время работы приложения теперь около 10 секунд, так как приложение ждет завершения первого дополнительного потока, а он как раз и работает 10с.

Измените значение свойства IsBackground для второго дополнительного потока на false и снова выполните приложение. Обратите внимание на общее время работы приложения.



+- 20 сек: общее время работы приложения теперь около 20 секунд, так как приложение будет ожидать завершения обоих дополнительных потоков.

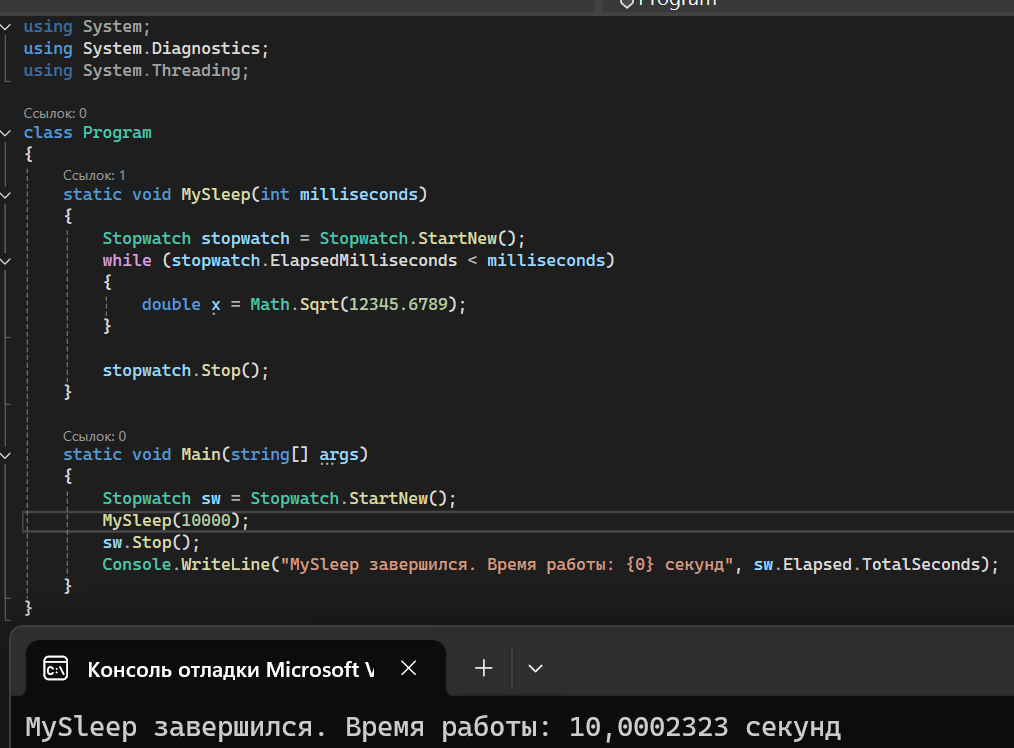
При IsBackground = true приложение может завершиться раньше, если главный поток заканчивает работу, и все фоновые потоки останавливаются.

При IsBackground = false приложение будет продолжать работать до тех пор, пока все рабочие потоки не завершат свою работу.

Можно еще использовать join

**Задание 05 (вспомогательное)**

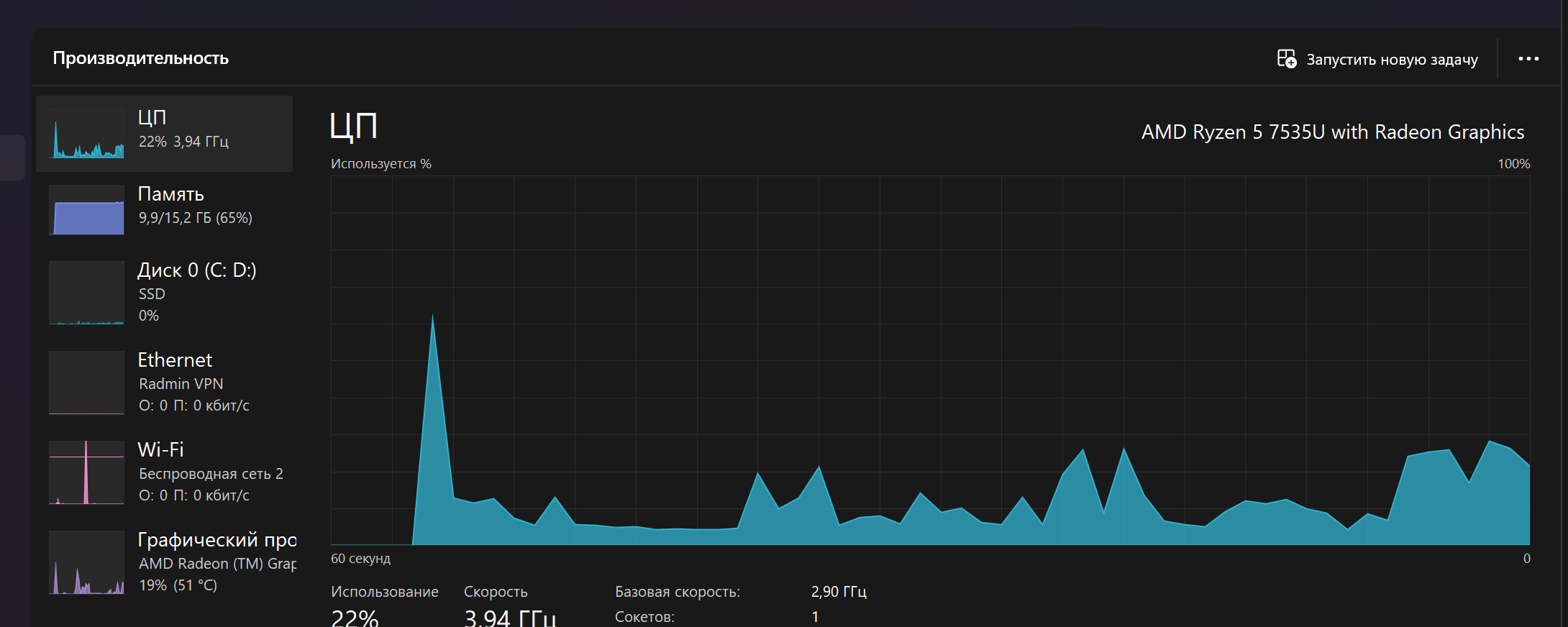
Создайте функцию, которая производит ЛЮБЫЕ вычисления длительностью n миллисекунд на вашем компьютере (для последующих заданий метод Thread. Sleep(n) не подходит, так как он освобождает центральный процессор и ничего не делает) . Убедитесь, что MySleep(10000) работает ровно 10 секунд.



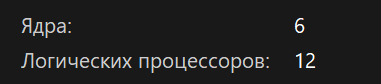
До запуска:



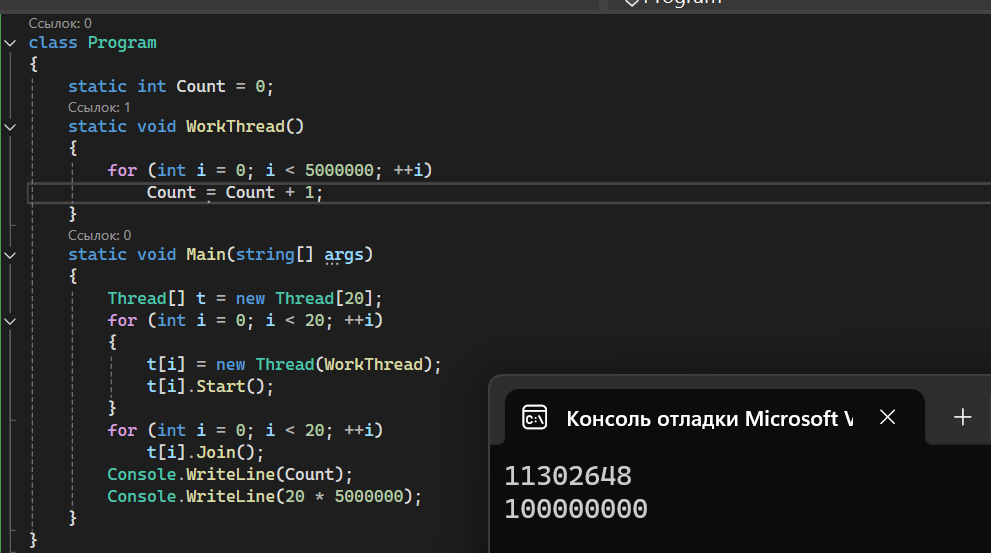
После:



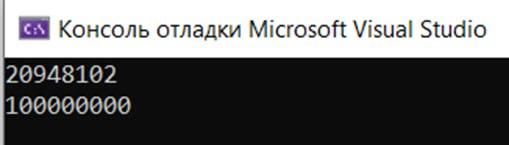
Узнайте количество ядер и логических процессоров в вашем компьютере

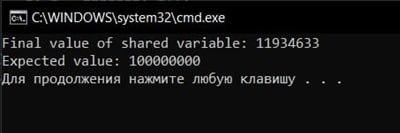


**Задание 06**  
Разработайте консольное приложение **OS04\_06** на языке С#, запускающее 20 потоков, каждый из которых в цикле 5000000 раз увеличивает на единицу значение общей для всех потоков переменной. Исходное значение переменной — ноль. Выведите результат и сравните с произведением  
20х5000000.



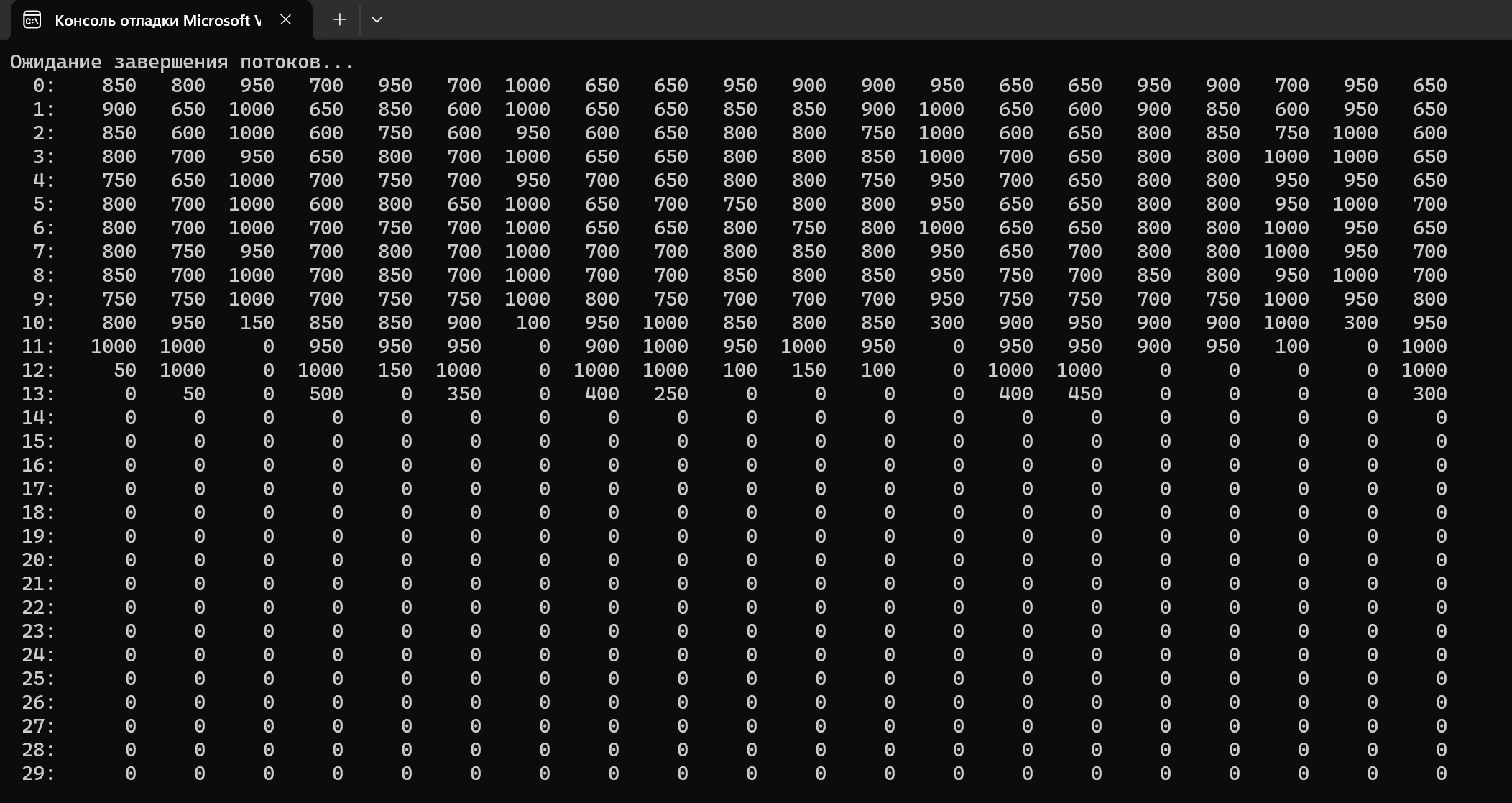
Разница между ожидаемым и фактическим результатом в программе возникает из-за **гонки данных**. Это происходит, когда несколько потоков одновременно обращаются к одной и той же переменной без синхронизации Когда поток читает значение переменной, увеличивает его и записывает обратно, другой поток может параллельно сделать то же самое. В результате часть операций теряется, и итоговое значение оказывается меньше, чем ожидаемое.

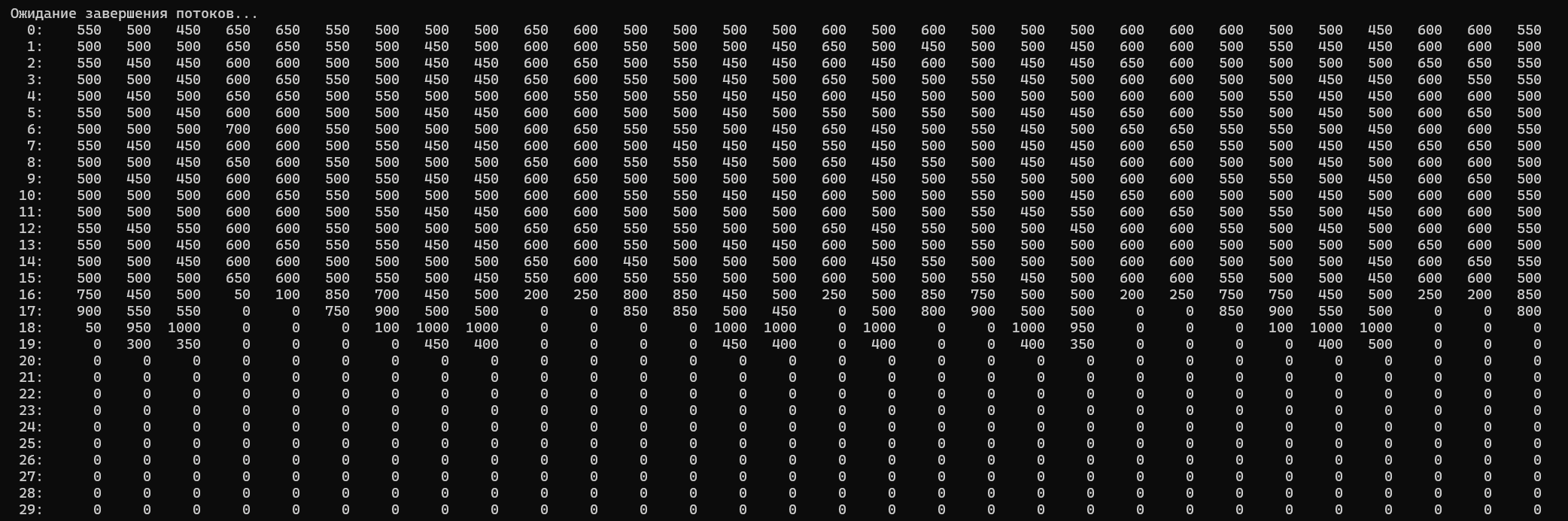




**Задание 07**

Разработайте консольное приложение OS04\_07 на языке С#, запускающее N потоков, каждый из которых будет производить вычисления t секунд (использовать разработанный в задании 5 метод) , используя класс System.Threading.Thread. Сохраните информацию о работе потоков в течение T секунд и выведите на экран в виде таблицы < >. Подберите подходящие параметры в зависимости от количества логических процессоров в вашем компьютере (например, для четырех логических процессоров N = 10, t = 10, T=30).





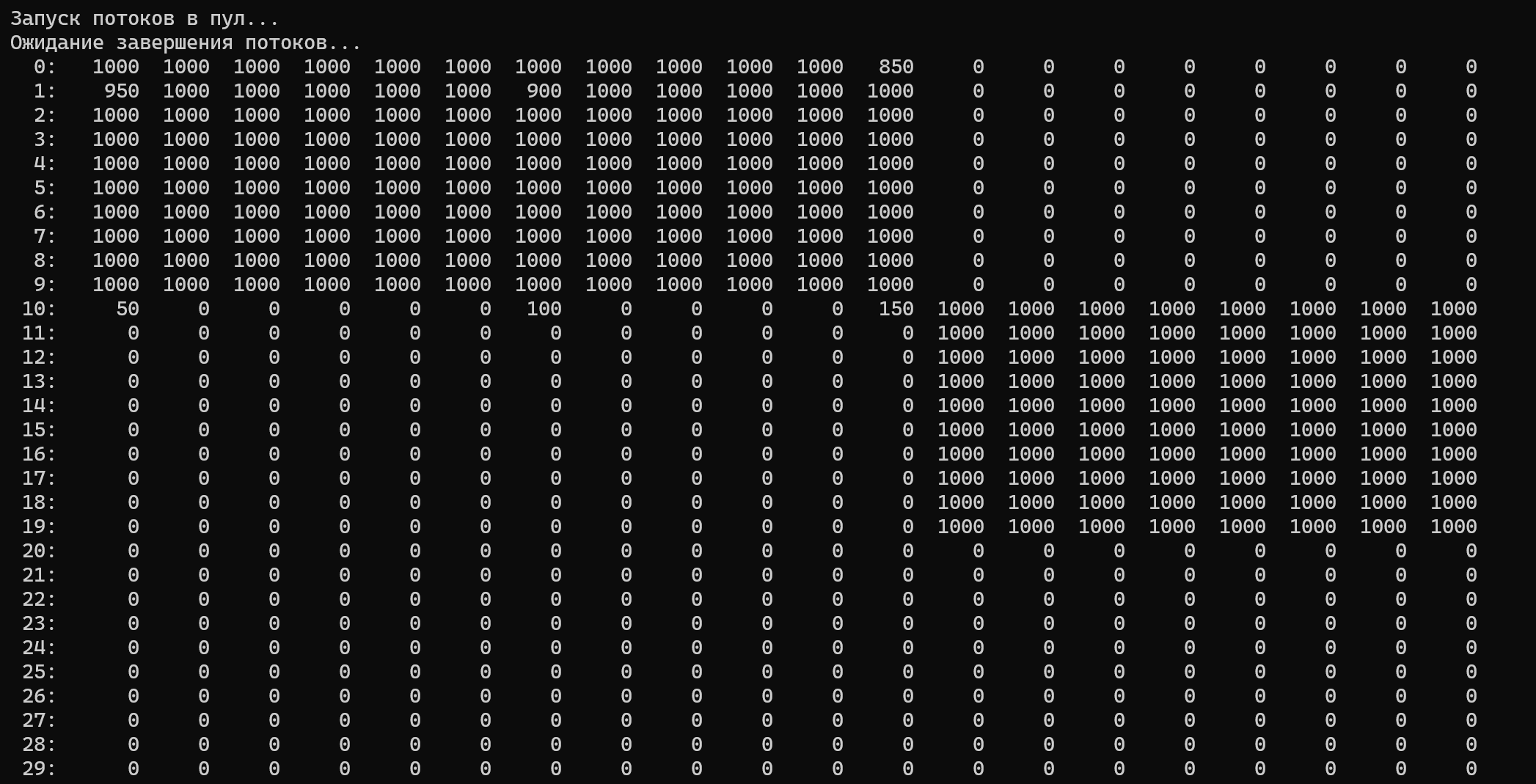
Строки – время

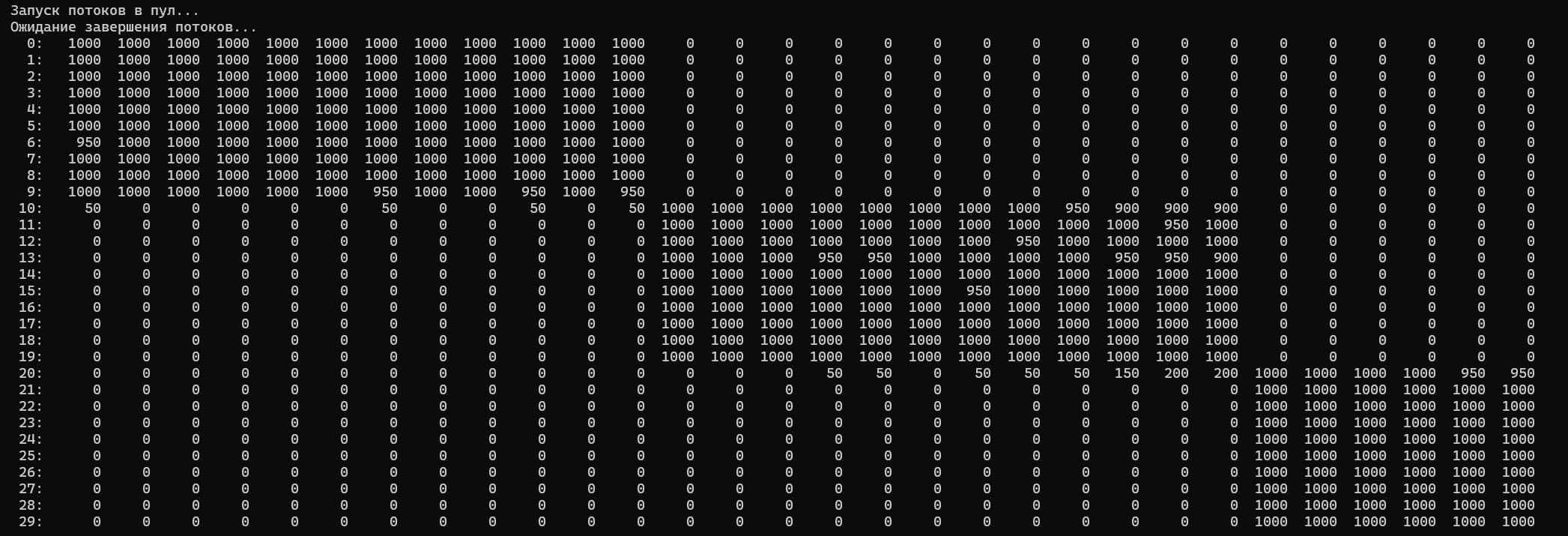
Столбцы – потоки

Числа – единицы работы, выполненные потоком. Эти значения могут варьироваться из-за конкуренции за ресурсы, при этом потоки могут блокироваться, если другие потоки занимают ресурсы. Если 0 – значит поток завершил свою работу.

**Задание 08**

Скопируйте консольное приложение OS04\_07 как OS04\_08. Теперь используйте пул потоков. Выведите статистику работы потоков на экран в виде таблицы .

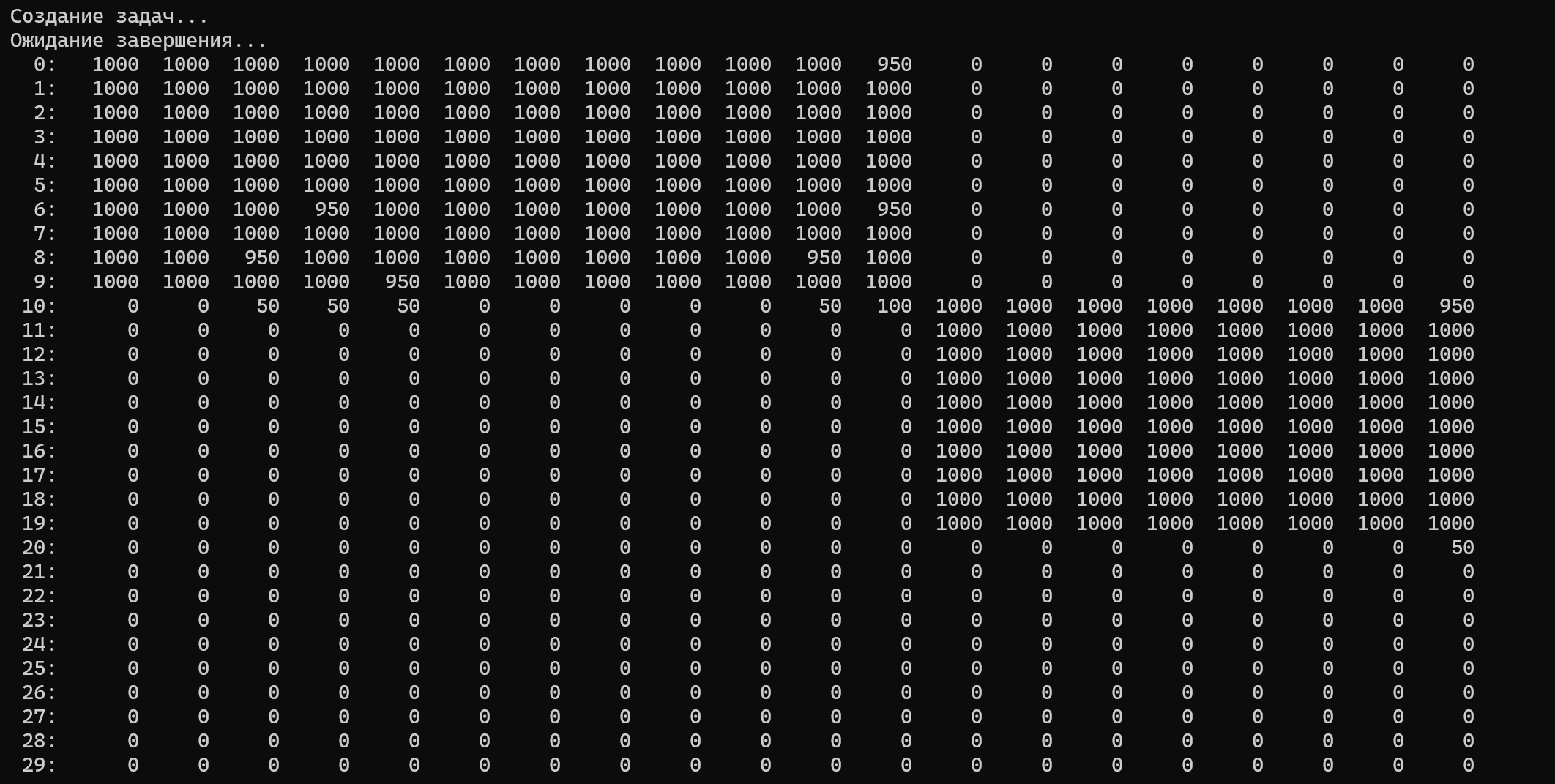


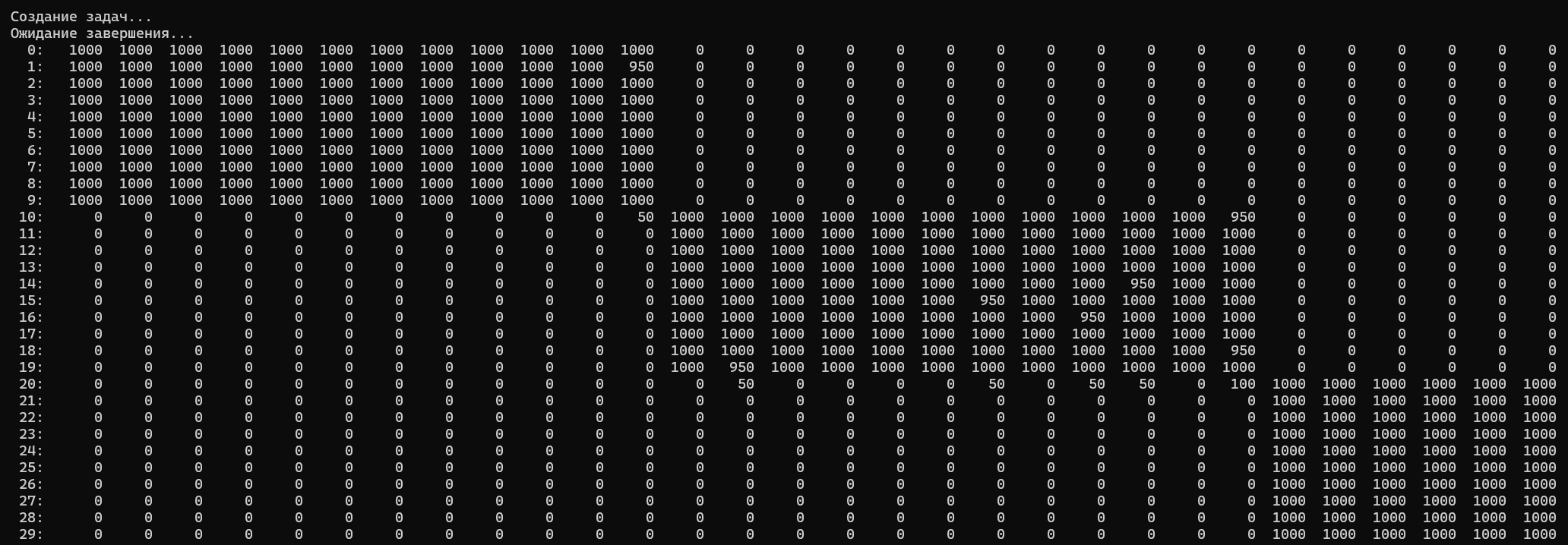


Пул потоков может создает определенное количество потоков (тут сначала 12, а потом остальные), которые начинают выполняться. Эти 12 потоков работают и завершаются, а затем пул может создает или дополнительные потоки для выполнения оставшейся работы. В результате, мы видим, что сначала активируется одна половина потоков, а затем другая. По сути при использовании пула потоков, достигается максимальная производительность для каждого.

**Задание 09**

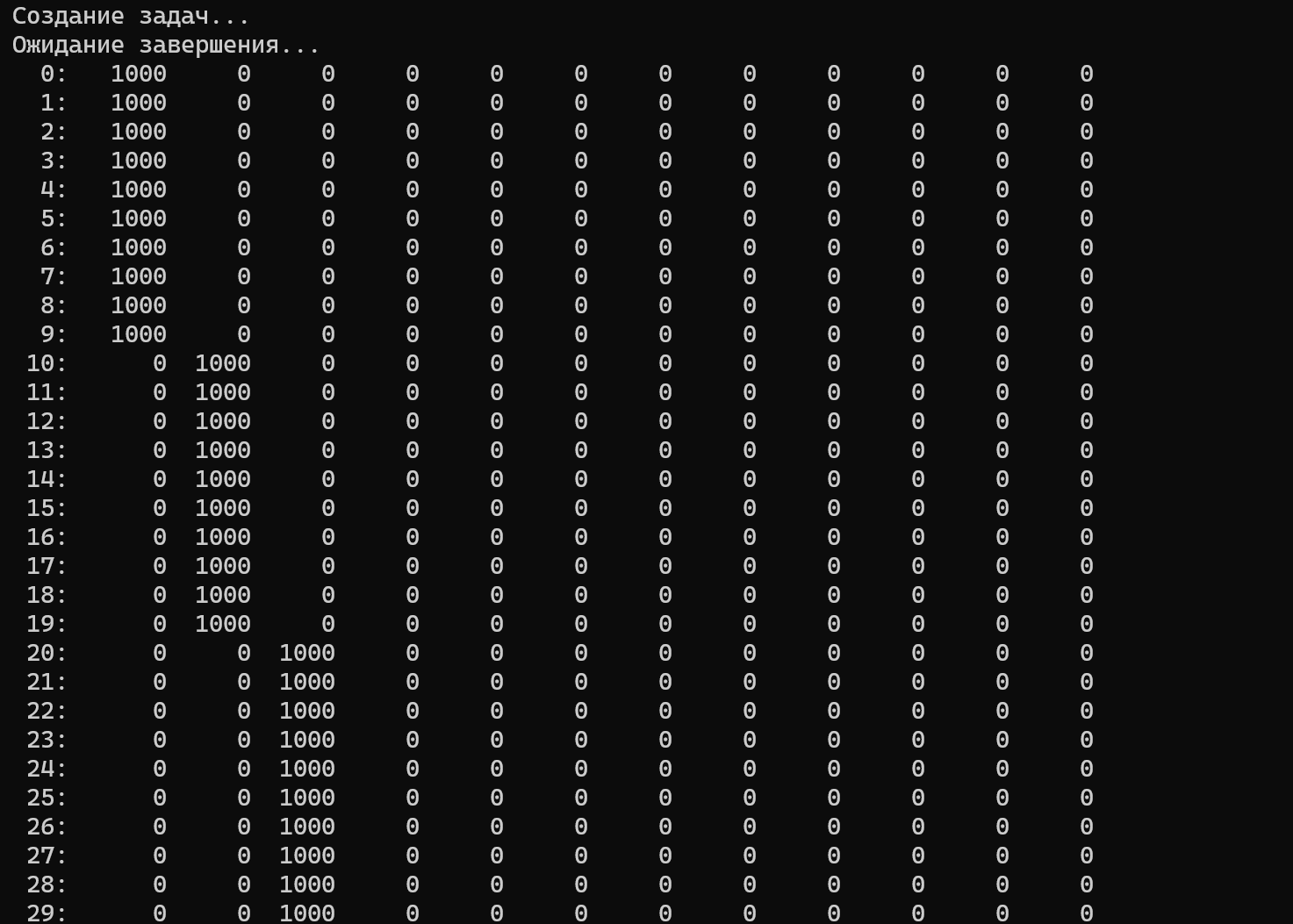
Скопируйте консольное приложение OS04\_07 как OS04\_09. На этот раз используйте System.Threading.Tasks. Task. Выведите статистику работы потоков на экран в виде таблицы.





Задачи используют пул потоков (Thread Pool), что позволяет избежать накладных расходов на создание и уничтожение потоков. Когда задача завершена, поток возвращается в пул, чтобы быть повторно использованным для других задач. Так как задачи автоматически используют потоки из пула, система не перегружена большим количеством одновременно работающих потоков, что может привести к уменьшению производительности.

**Задание 10**  
Скопируйте консольное приложение **OS04\_09** как **OS04\_10**. Уменьшите количество задач до количества логических процессоров. Организуйте выполнение задач по очереди. Выведите статистику работы потоков на экран в виде таблицы

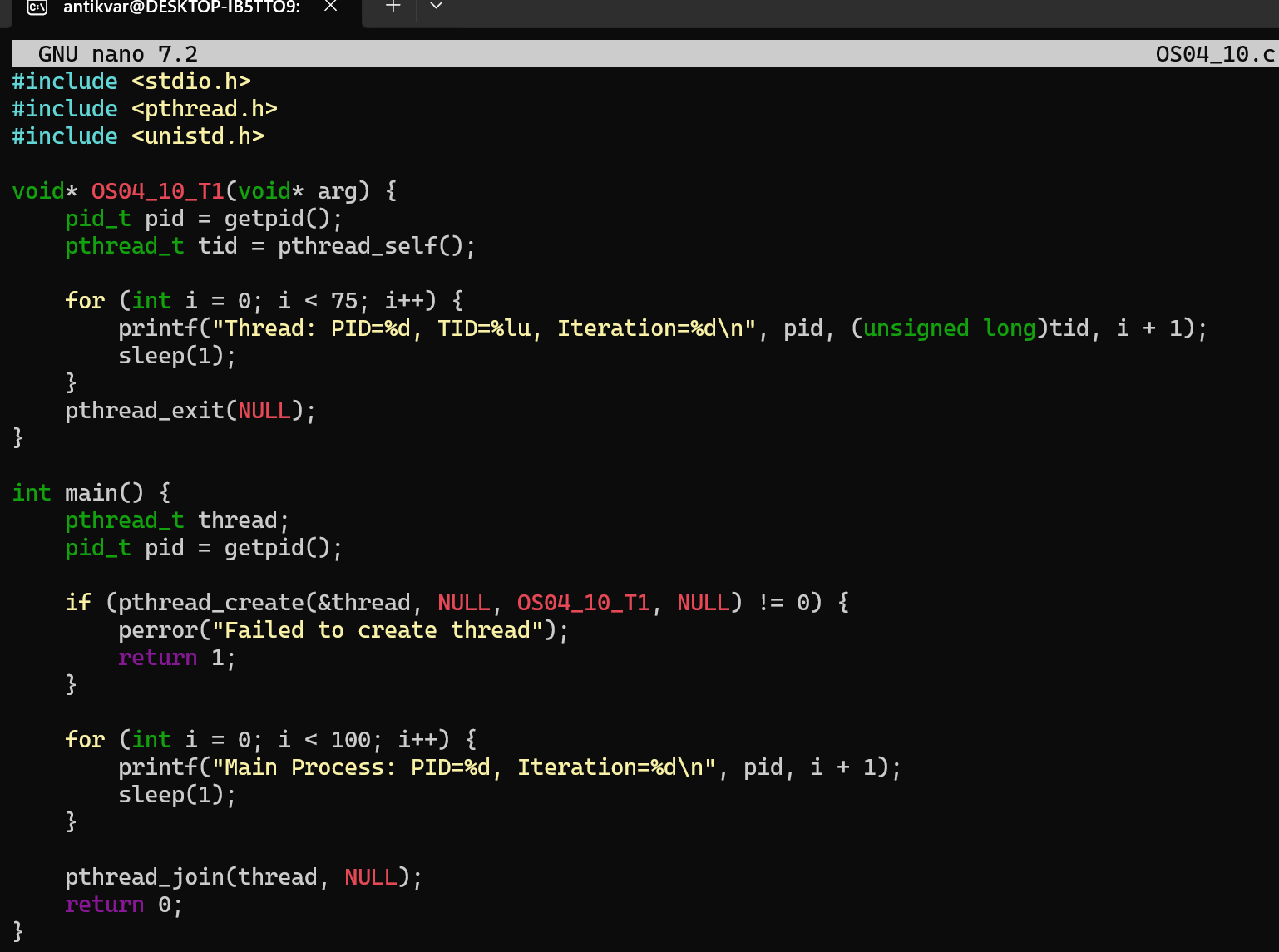


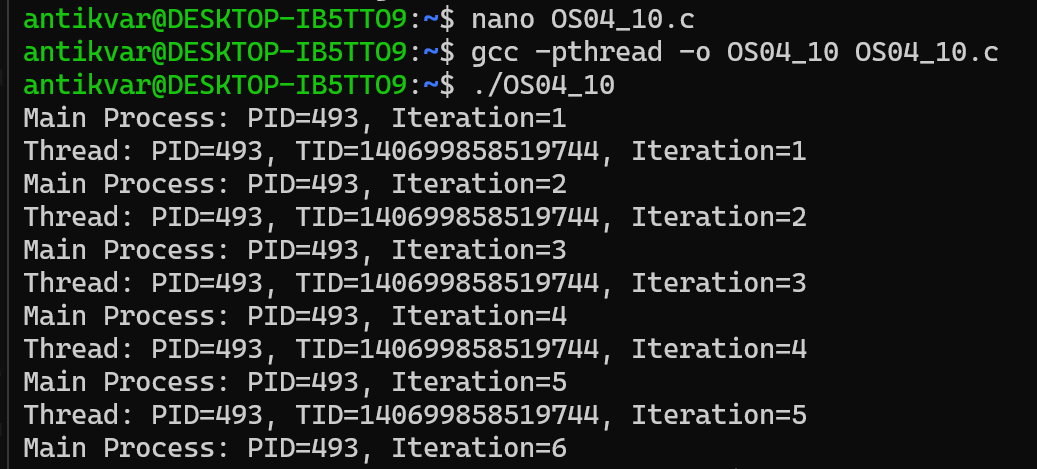
Мы запускаем задачи последовательно с помощью ContinueWith, это означает, что каждая задача ожидает завершения предыдущей перед тем, как начать выполнение. В результате только одна задача выполняется в данный момент времени, и только она обновляет значения в матрице, пока не завершится, и только потом начинает следующая задача.

**Задание 11**

Разработайте на языке консольное Linux-приложение OS04\_10 на языке С, выполняющее цикл 100 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса (использовать функции из pthread.h).

Процесс OS04\_10 должен создать поток: потоковая функция OS04\_10\_T1. Поток OS04\_10\_T1 - выполняет цикл 75 итераций с временной задержкой в 1 сек. с выводом на консоль идентификаторов процесса.

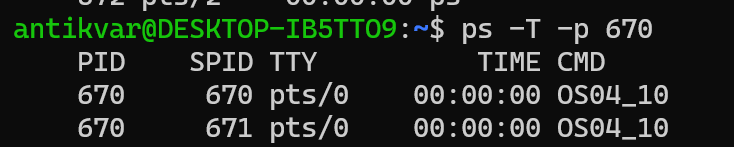




Продемонстрируйте информацию о потоках процесса OS04\_10 с помощью утилиты ps.

Ps –e – все процессы на сервере

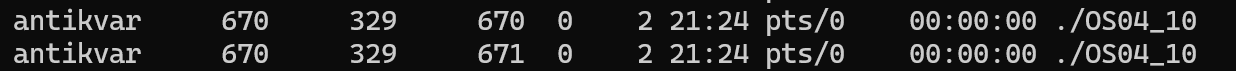
ps -T –p (пид процесса) – показать потоки процесса



ps –eLf

-L — выводит информацию о потоках.

-f — полный формат вывода, включая имя пользователя, PID, PPID, время запуска и команду.



**Задание 12.** Ответьте на следующие вопросы

**Что такое поток управления OS?**

Поток выполнения (англ. thread — нить) — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов.

В частности, потоки выполнения разделяют:

* Инструкции процесса (его код);
* Контекст процесса (значения переменных, которые они имеют в любой момент времени).

**С помощью каких системных вызовов создаются потоки в Windows и Linux?**

Новый поток создается с помощью вызова функции **pthread\_create**. В качестве значения функции возвращается идентификатор только что созданного потока. Поток создается путем выполнения start\_routine с arg в качестве единственного аргумента. Если start\_routine возвращает значение, эффект такой же, как если бы был неявный вызов pthread\_exit() с использованием возвращаемого значения start\_routine в качестве статуса выхода.

* pthread\_t \*thread: Указатель на переменную для хранения идентификатора потока.
* const pthread\_attr\_t \*attr: Указатель на атрибуты потока, передайте NULL для настроек по умолчанию.
* void \*(\*start\_routine)(void \*): Указатель на функцию, которую будет выполнять поток.
* void \*arg: Аргумент, передаваемый в функцию потока (может быть NULL)

Функция **CreateThread** создает новый поток для процесса. Создаваемый поток должен указать начальный адрес кода, который должен выполнить новый поток. Как правило, начальный адрес — это имя функции, определенной в коде программы. Эта функция принимает один параметр и возвращает значение DWORD.

HANDLE CreateThread(

LPSECURITY\_ATTRIBUTES lpThreadAttributes, // атрибуты безопасности

SIZE\_T dwStackSize, // размер стека потока

LPTHREAD\_START\_ROUTINE lpStartAddress, // указатель на функцию, которая будет выполняться в потоке

LPVOID lpParameter, // параметр, который передается в поток

DWORD dwCreationFlags, // флаги создания потока

LPDWORD lpThreadId // указатель на переменную, которая получит идентификатор потока

);

**Что такое системные и пользовательские потоки?**

При первом подходе (на уровне пользовательского пространства) процесс ведет собственную таблицу потоков, а ядро считает этот процесс однопоточным. Переключение потоков в этом случае требует меньше затрат процессорного времени. Однако возникают проблемы с планированием — если поток не возвращает управление добровольно, другие потоки не смогут выполняться. Также системные вызовы или отсутствие страницы в памяти могут заблокировать весь процесс.

При реализации потоков на уровне ядра не требуется новых неблокирующих системных вызовов. Более того, если один из потоков сталкивается с отсутствующей страницей, ядро может проверить, есть ли у процесса другие готовые к выполнению потоки, и запустить один из них, пока ожидается загрузка страницы с диска. Главным недостатком этого подхода являются высокие затраты на системные вызовы, что особенно ощутимо при частых операциях с потоками, таких как создание или удаление. Управляются операционной системой через системные вызовы (например, pthread\_create в Linux или CreateThread)

Системный вызов — это механизм, с помощью которого программы в пользовательском пространстве взаимодействуют с ядром операционной системы для выполнения различных привилегированных операций

**Что такое многопоточность?**

Многопоточность позволяет процессу, созданному в операционной системе, состоять из нескольких потоков, которые выполняются «параллельно», то есть без строго заданного порядка. В .NET Framework начиная с версии 4 рекомендуется использовать библиотеку параллельных задач (TPL) и Parallel LINQ (PLINQ) для организации многопоточности.

TPL и PLINQ опираются на потоки пула потоков (ThreadPool). Класс System.Threading.ThreadPool предоставляет .NET-приложениям пул рабочих потоков, которые можно использовать для выполнения задач. Также можно напрямую работать с классом System.Threading.Thread, который представляет управляемый поток.

**Что такое контекст потока и для чего он нужен?**

Контекст потока содержит все сведения, позволяющие потоку безболезненно возобновить выполнение, в том числе набор регистров процессора и стек потока. Несколько потоков могут выполняться в контексте процесса. Все потоки процесса используют общий диапазон виртуальных адресов. Поток может исполнять любую часть программного кода, включая части, выполняемые в данный момент другим потоком

* Основные компоненты контекста потока:
* Регистры процессора
* Стек
* Состояние потока
* Идентификатор потока
* Приоритет потока
* Используемые ресурсы

**Перечислите состояния, в которых может быть поток и поясните их назначение.**

Каждый созданный в системе поток может находиться в одном из трех состояний:

* Состояние выполнения: в этом состоянии код потока выполняется процессором. На однопроцессорных платформах только один поток может находиться в этом состоянии в любой момент времени.
* Состояние готовности к выполнению: поток готов продолжить свою работу и ожидает освобождения центрального процессора (ЦП).
* Состояние ожидания: поток не требует времени ЦП и ждет наступления определенного события, например, завершения операции ввода/вывода или освобождения необходимого ресурса. Потоки в этом состоянии часто называют блокированными.

В состояниях готовности и ожидания может находиться несколько потоков, поэтому система создает отдельные списковые структуры для хранения их дескрипторов. Организация этих списков зависит от принципов, заложенных в систему планирования потоков для данной операционной системы.

**Что такое LWP?**

LWP - поддерживаемый ядром пользовательский поток

**Что такое потокобезопасность программного кода?**

Потокобезопасность — это предотвращение гонок данных — ситуаций, в которых данные устанавливаются в правильные или неправильные значения в зависимости от порядка, в котором несколько потоков получают доступ к данным и изменяют их. Процедура потокобезопасна, когда она логически корректна при одновременном выполнении несколькими потоками.

**Что такое реентерабельность кода?**

Компьютерная программа в целом или её отдельная процедура называется реентерабельной, если она разработана таким образом, что одна и та же копия инструкций программы в памяти может быть совместно использована несколькими пользователями или процессами [3]. При этом второй пользователь может вызвать реентерабельный код до того, как с ним завершит работу первый пользователь и это как минимум не должно привести к ошибке, а при корректной реализации не должно вызвать потери вычислений (то есть не должно появиться необходимости выполнять уже выполненные фрагменты кода).

**Что такое Fiber?**

Волокно — наименьшая последовательность запрограммированных инструкций, которой можно управлять в пользовательском режиме, внешнем по отношению к планировщику операционной системы. Волокно представляет память для стека выполнения и набор регистров ЦП, необходимых для выполнения инструкций. Волокна могут выполняться только явно потоками и никогда не будут запланированы планировщиком операционной системы. Запуск волокна является кооперативным, то есть поток должен явно выбрать волокно для запуска, а волокно должно сообщить потоку, в котором оно запущено, что оно завершено или его необходимо приостановить.

Волокно похоже на поток в том смысле, что у него есть свой стек, но в отличие от потока в том, что операционная система никогда не планирует волокно. Вместо этого рабочие потоки carb.tasking запускают волокна по мере их доступности.