Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Операционные системы

Студент: Лопатнюк П.В.

ФИТ 3 курс 1 группа

Преподаватель: Бернацкий П.В.

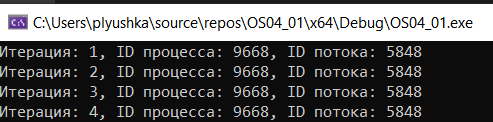
Минск 2024

**Лабораторная работа 04**

**Задание 01.**

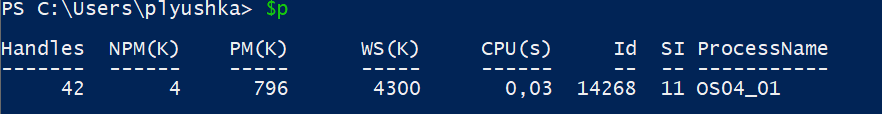
1. Разработайте консольное Windows-приложение **OS04\_01** на языке С++, выполняющее длинный цикл с временной задержкой и с выводом на консоль идентификаторов текущего процесса и текущего потока

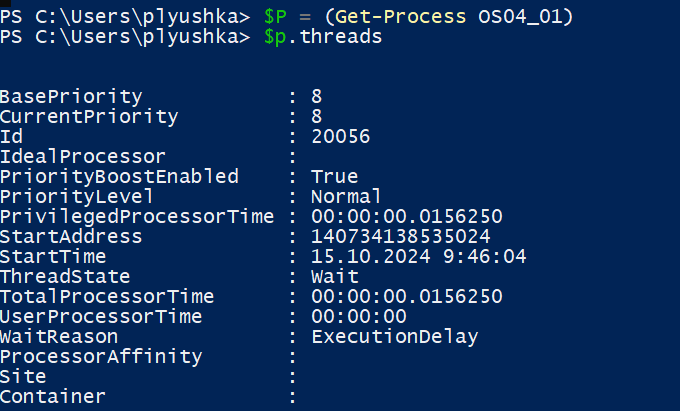
|  |
| --- |
| #include <windows.h>  #include <iostream>  #include <thread>  using namespace std;  int main()  {  setlocale(LC\_ALL, "rus");  DWORD processId = GetCurrentProcessId();  DWORD threadId = GetCurrentThreadId();  for (int i = 0; i < 1000; ++i)  {  cout << "Итерация: " << i + 1 << ", ID процесса: " << processId << ", ID потока: " << threadId << endl;  Sleep(1000);  }  return 0;  } |



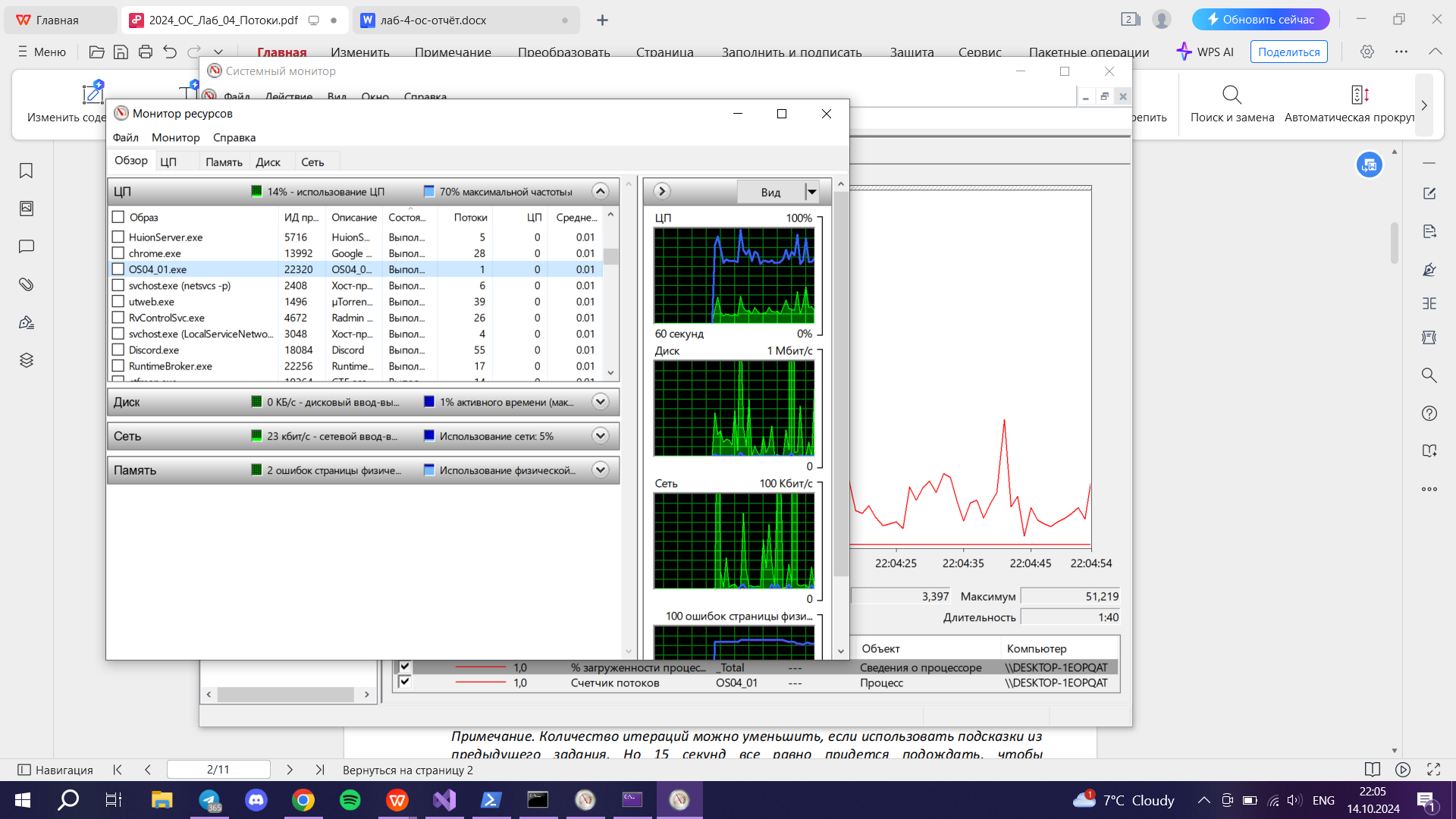
2. Продемонстрируйте информацию об потоках процесса **OS04\_01** с помощью утилит **PowerShell ISE** и **Performance Monitor**.

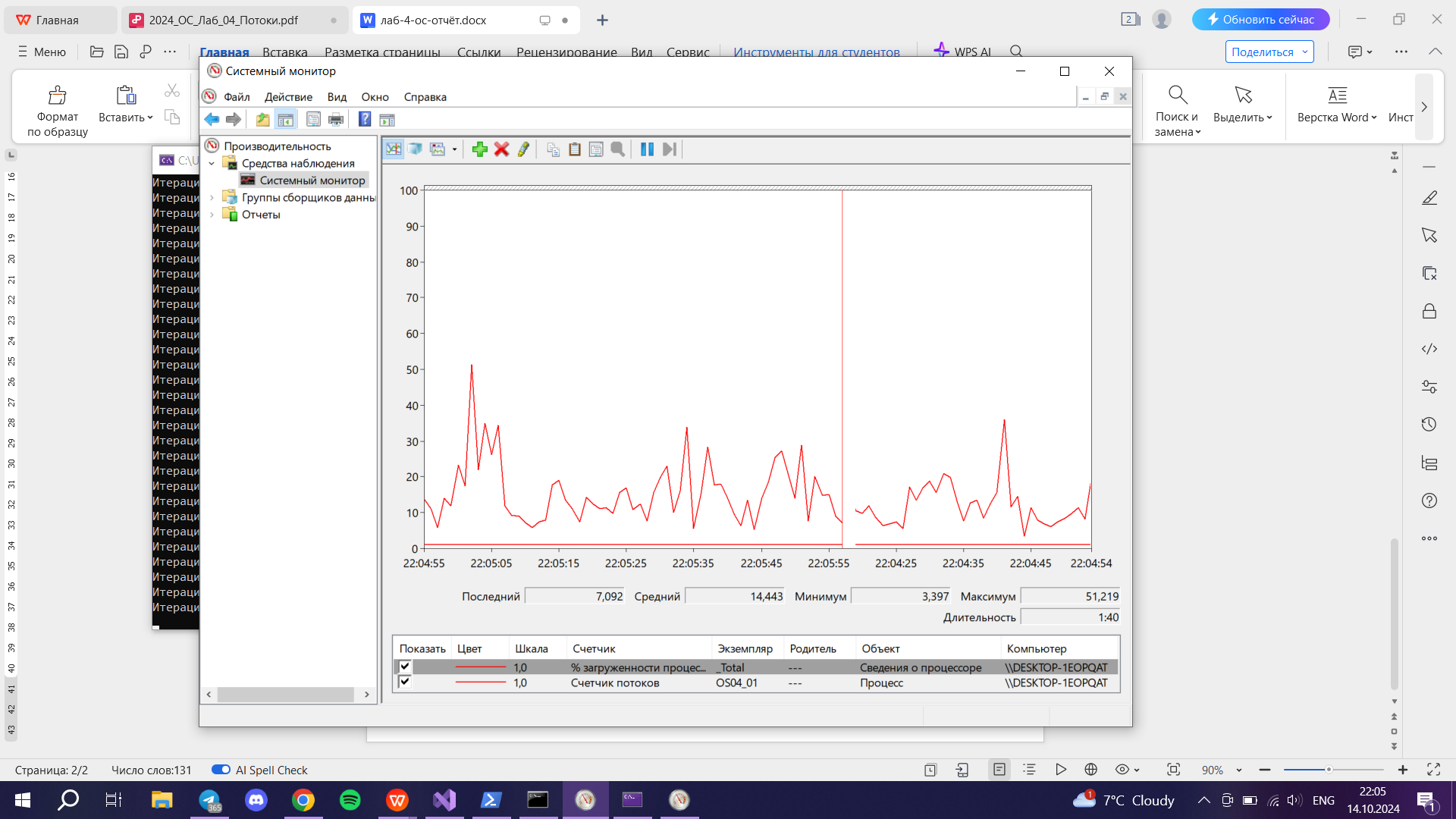
**PowerShell ISE**





**Performance Monitor**





**Задание 02.**

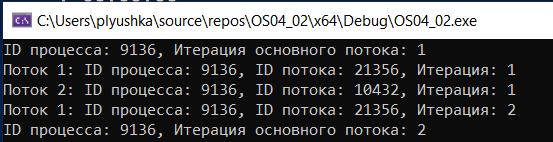
1. Разработайте консольное Windows-приложение **OS04\_02** на языке С++, выполняющее цикл 100 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса.

4. Процесс **OS04\_02** должен создать два потока: потоковые функции **OS04\_02\_T1, OS04\_02\_T2.**

5. Поток **OS04\_02\_T1** - выполняет цикл 50 итераций с временной задержкой в 1 сек. с выводом на консоль идентификаторов процесса и потока.

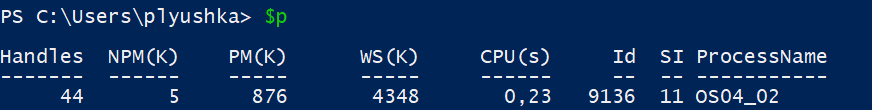
6. Поток **OS04\_02\_T2** - выполняет цикл 125 итераций с временной задержкой в 1 сек. с выводом на консоль идентификаторов процесса и потока.

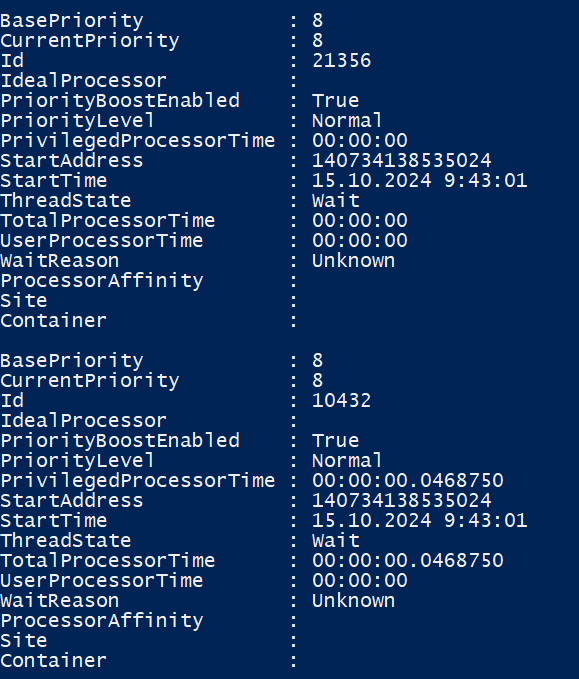
|  |
| --- |
| #include <iostream>  #include <windows.h>  using namespace std;  CRITICAL\_SECTION criticalSection;  DWORD WINAPI OS04\_02\_T1(LPVOID) {  DWORD processId = GetCurrentProcessId();  DWORD threadId = GetCurrentThreadId();  for (int i = 0; i < 50; ++i) {  EnterCriticalSection(&criticalSection);  cout << "Поток 1: ID процесса: " << processId << ", ID потока: " << threadId  << ", Итерация: " << i + 1 << endl;  LeaveCriticalSection(&criticalSection);  Sleep(1000);  }  return 0;  }  DWORD WINAPI OS04\_02\_T2(LPVOID) {  DWORD processId = GetCurrentProcessId();  DWORD threadId = GetCurrentThreadId();  for (int i = 0; i < 125; ++i) {  EnterCriticalSection(&criticalSection);  cout << "Поток 2: ID процесса: " << processId << ", ID потока: " << threadId  << ", Итерация: " << i + 1 << endl;  LeaveCriticalSection(&criticalSection);  Sleep(1000);  }  return 0;  }  int main() {  setlocale(LC\_ALL, "rus");  DWORD processId = GetCurrentProcessId();  InitializeCriticalSection(&criticalSection);  HANDLE hThread1 = CreateThread(NULL, 0, OS04\_02\_T1, NULL, 0, NULL);  HANDLE hThread2 = CreateThread(NULL, 0, OS04\_02\_T2, NULL, 0, NULL);  for (int i = 0; i < 100; ++i) {  EnterCriticalSection(&criticalSection);  cout << "ID процесса: " << processId << ", Итерация основного потока: " << i + 1 << endl;  LeaveCriticalSection(&criticalSection);  Sleep(1000);  }  if (hThread1 == NULL || hThread2 == NULL) {  cerr << "Ошибка при создании потоков." << endl;  return 1;  }  WaitForSingleObject(hThread1, INFINITE);  WaitForSingleObject(hThread2, INFINITE);  CloseHandle(hThread1);  CloseHandle(hThread2);  DeleteCriticalSection(&criticalSection);  return 0;  } |



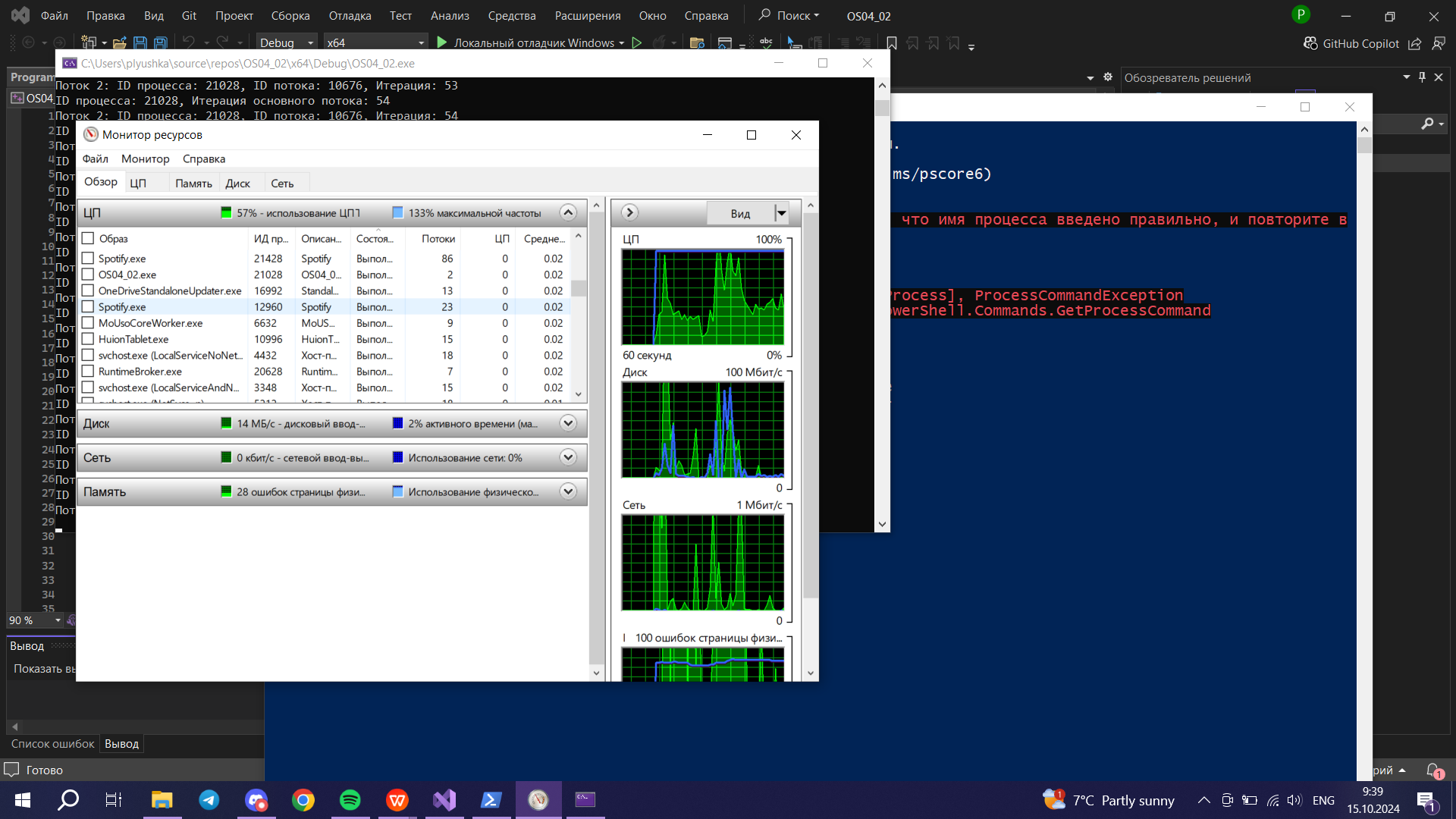
7. Продемонстрируйте информацию об потоках процесса **OS04\_02** с помощью утилит **PowerShell ISE** и **Performance Monitor**.

**PowerShell**





**Performance Monitor**.

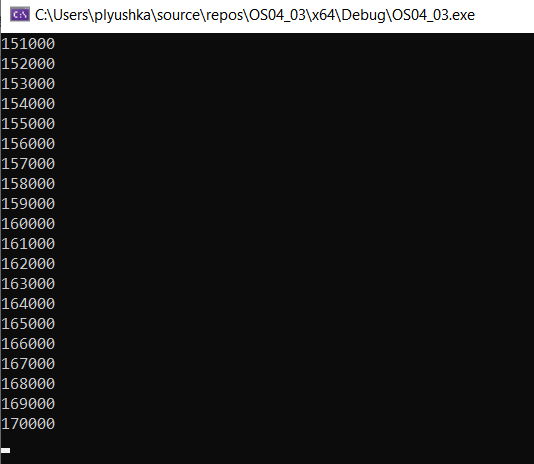


**Задание 03.**

8. Определите, какое максимальное количество потоков можно создать в одном процессе на вашем компьютере, используя нижеприведенный программный код (или аналогичный).

Сравните результаты между собой. От чего зависит максимальное количество потоков? Какое ограничение накладывает операционная система.

|  |
| --- |
| #include <Windows.h>  #include <iostream>  DWORD WINAPI threadProc(LPVOID);  int main()  {  HANDLE hThread;  DWORD dwThreadId;  for (int Count = 0; Count < 1000000; Count++)  {  hThread = CreateThread(  NULL, 0, threadProc, NULL, CREATE\_SUSPENDED, &dwThreadId);  if (hThread == INVALID\_HANDLE\_VALUE || hThread == NULL) {  printf("CreateThread failed (error %d) after %d threads\n",  GetLastError(), Count);  break;  }  ResumeThread(hThread);  CloseHandle(hThread);  if (Count % 1000 == 0)  printf("%d\n", Count);  }  printf("Thread Main exited\n");  return 0;  }  DWORD WINAPI threadProc(LPVOID args)  {  Sleep(600000);  return 0;  } |

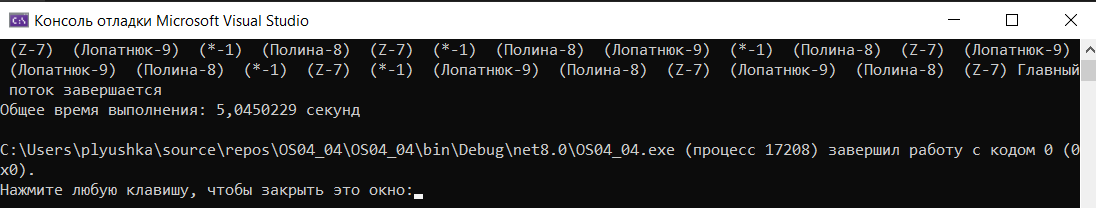


**Задание 04.**

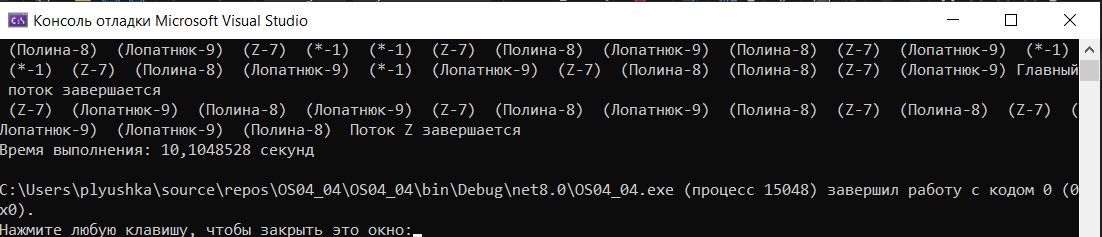
9. Создайте консольное приложение на C#, которое запускает три дополнительных потока и завершается через пять секунд. Один дополнительный поток завершается через 10 секунд, остальные два – через двадцать секунд.

|  |
| --- |
| using System.Diagnostics;  class Program  {  private static Stopwatch time;  static void ThreadZed()  {  for (int i = 0; i < 10; i++)  {  Console.Write(" (Z-{0}) ", Thread.CurrentThread.ManagedThreadId);  Thread.Sleep(1000);  }  Console.WriteLine(" Поток Z завершается ");  }  static void ThreadWithParam(object o)  {  for (int i = 0; i < 20; i++)  {  Console.Write(" ({0}-{1}) ", o.ToString(),  Thread.CurrentThread.ManagedThreadId);  Thread.Sleep(1000);  }  }  static void Main(string[] args)  {  time = Stopwatch.StartNew();  var t1 = new Thread(ThreadZed);  var t1a = new Thread(ThreadWithParam);  var t1b = new Thread(ThreadWithParam);  t1.IsBackground = true; // false для п.11  t1a.IsBackground = true; // false для п.12  t1b.IsBackground = true;  t1.Start();  t1a.Start("Полина");  t1b.Start("Лопатнюк");    for (int i = 0; i < 5; i++)  {  Console.Write(" (\*-{0}) ", Thread.CurrentThread.ManagedThreadId);  Thread.Sleep(1000);  }  Console.WriteLine("Главный поток завершается");  time.Stop();  Console.WriteLine("Общее время выполнения: {0} секунд", time.Elapsed.TotalSeconds);  }  } |

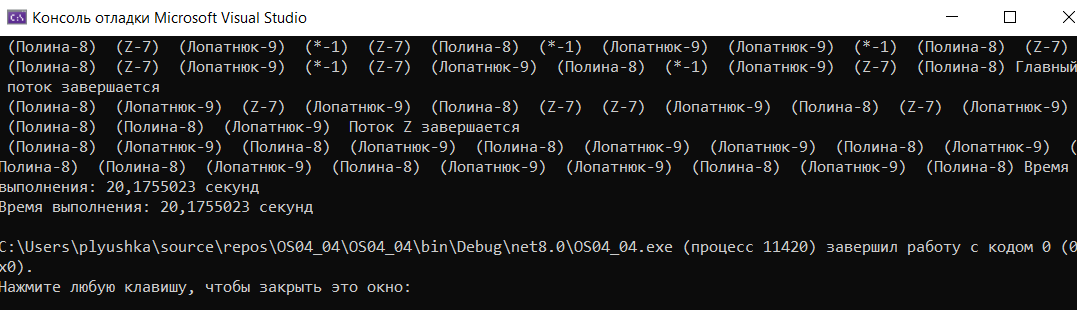
10. Выполните приложение. Обратите внимание на общее время работы приложения.



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



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

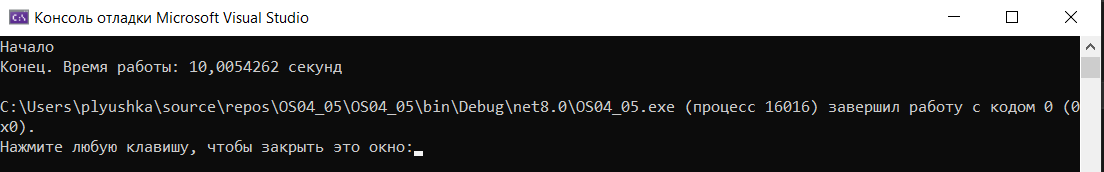


13. Вставьте три скриншота в отчет и объясните результат.

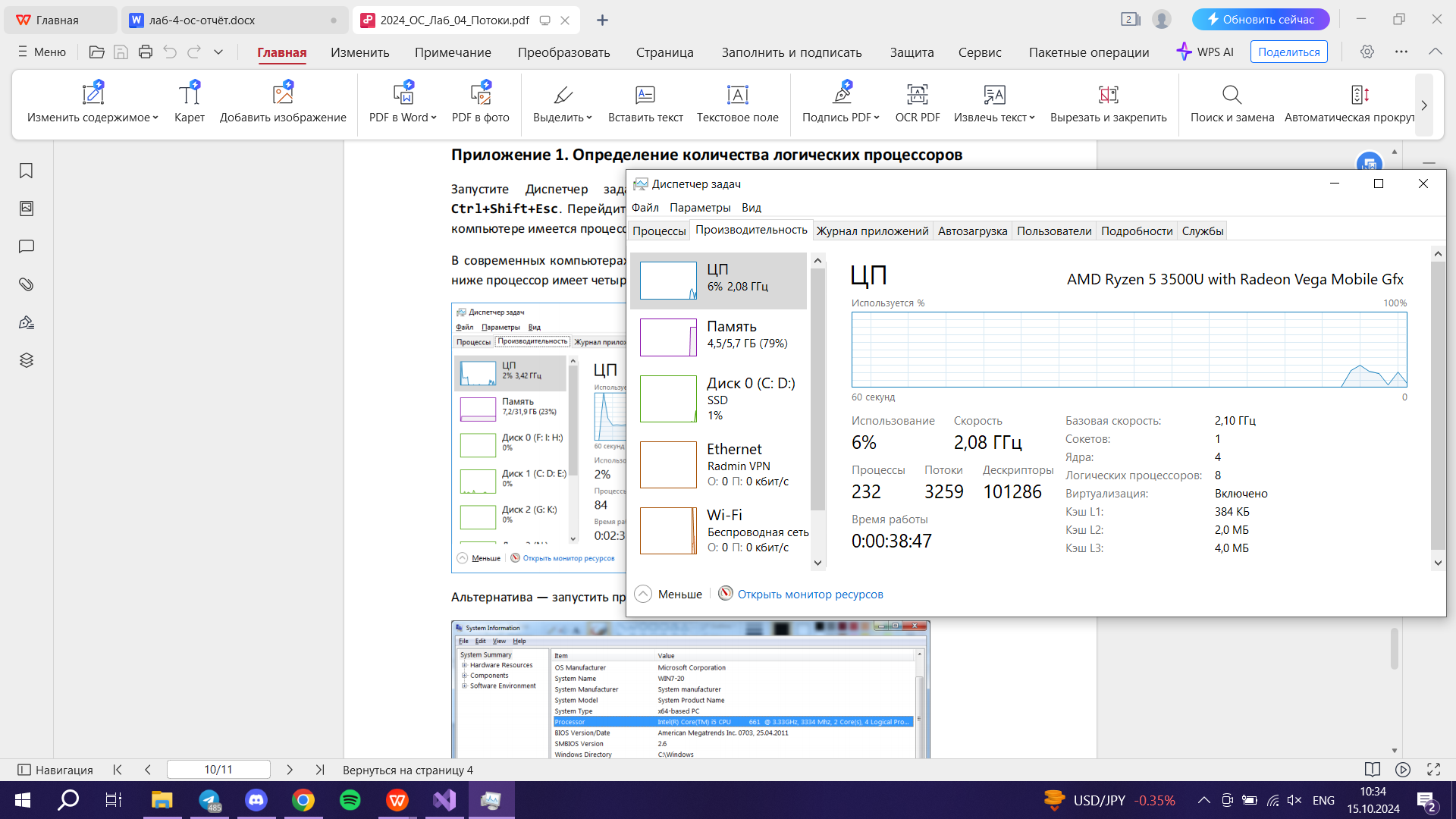
**Задание 05 (вспомогательное).**

15. 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.

|  |
| --- |
| using System.Diagnostics;  public class Program  {  public static void MySleep(int milliseconds)  {  var startTime = DateTime.Now;  while (DateTime.Now - startTime < TimeSpan.FromMilliseconds(milliseconds))  {  int result = 1;  for (int i = 0; i < 10000; i++)  {  result \*= 2;  }  }  }  public static void Main(string[] args)  {  Stopwatch time= Stopwatch.StartNew();  Console.WriteLine("Начало");  MySleep(10000);  time.Stop();  Console.WriteLine("Конец. Время работы: {0} секунд", time.Elapsed.TotalSeconds);  }  } |



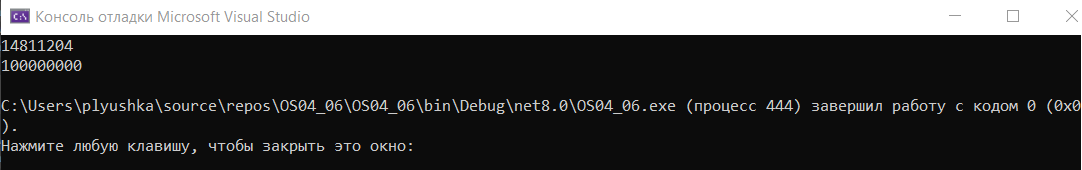
16. Узнайте количество ядер и логических процессоров в вашем компьютере (приложение 1).



**Задание 06.**

17. 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.

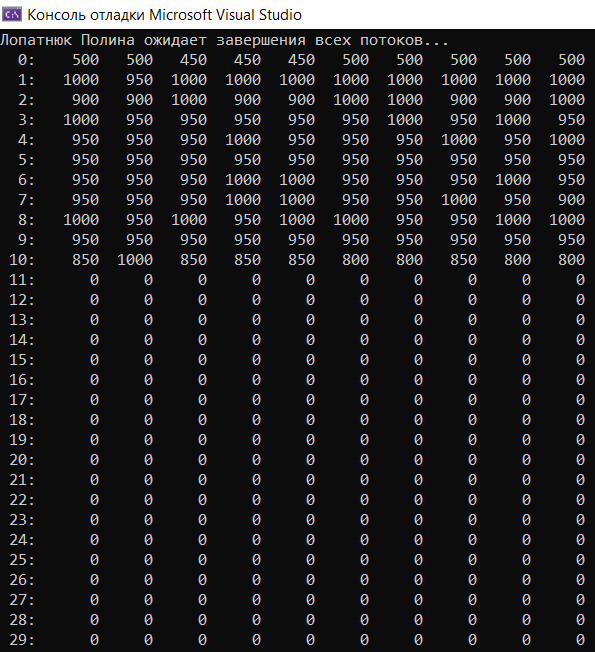
|  |
| --- |
| class Program  {  static int Count = 0;  static void WorkThread()  {  for (int i = 0; i < 5000000; ++i)  Count = Count + 1;  }  static void Main(string[] args)  {  Thread[] t = new Thread[20];  for (int i = 0; i < 20; ++i)  {  t[i] = new Thread(WorkThread);  t[i].Start();  }  for (int i = 0; i < 20; ++i)  t[i].Join();  Console.WriteLine(Count);  Console.WriteLine(20 \* 5000000);  }  } |



**Задание 07.**

1. 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).

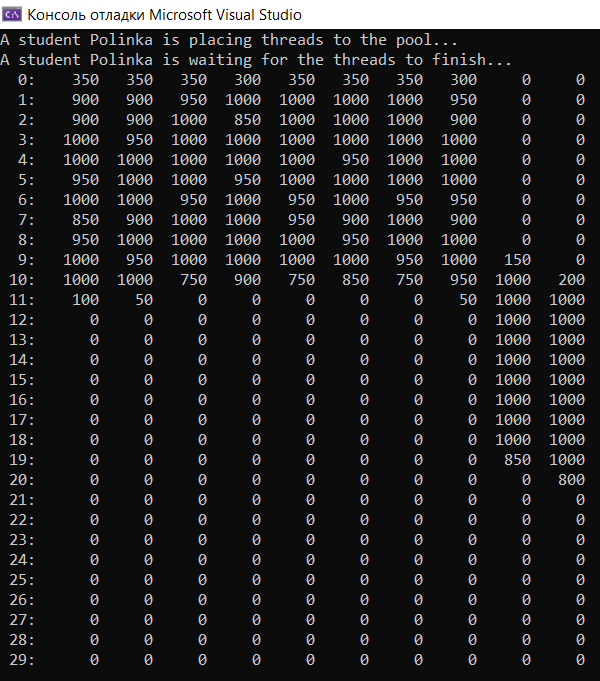
|  |
| --- |
| using System;  using System.Threading;  public class Program  {  const int ThreadCount = 10; // Количество потоков  const int ThreadLifeTime = 10; // Время работы каждого потока в секундах  const int ObservationTime = 30; // Время наблюдения в секундах  static int[,] Matrix = new int[ThreadCount, ObservationTime]; // Матрица для хранения данных о работе потоков  static DateTime StartTime = DateTime.Now;  static void WorkThread(object o)  {  int id = (int)o;  for (int i = 0; i < ThreadLifeTime \* 20; i++)  {  DateTime CurrentTime = DateTime.Now;  int ElapsedSeconds = (int)Math.Round(CurrentTime.Subtract(StartTime).TotalSeconds);  if (ElapsedSeconds < ObservationTime)  {  Matrix[id, ElapsedSeconds] += 50;  }  MySleep(50);  }  }  public static void MySleep(int milliseconds)  {  var startTime = DateTime.Now;  while (DateTime.Now - startTime < TimeSpan.FromMilliseconds(milliseconds))  {  int result = 1;  for (int i = 0; i < 10000; i++)  {  result \*= 2;  }  }  }  static void Main(string[] args)  {  Thread[] threads = new Thread[ThreadCount];  for (int i = 0; i < ThreadCount; ++i)  {  object o = i;  threads[i] = new Thread(WorkThread);  threads[i].Start(o);  }  Console.WriteLine("Лопатнюк Полина ожидает завершения всех потоков...");  for (int i = 0; i < ThreadCount; ++i)  {  threads[i].Join();  }  for (int s = 0; s < ObservationTime; s++)  {  Console.Write("{0,3}: ", s);  for (int th = 0; th < ThreadCount; th++)  {  Console.Write(" {0,5}", Matrix[th, s]);  }  Console.WriteLine();  }  }  } |



**Задание 08.**

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

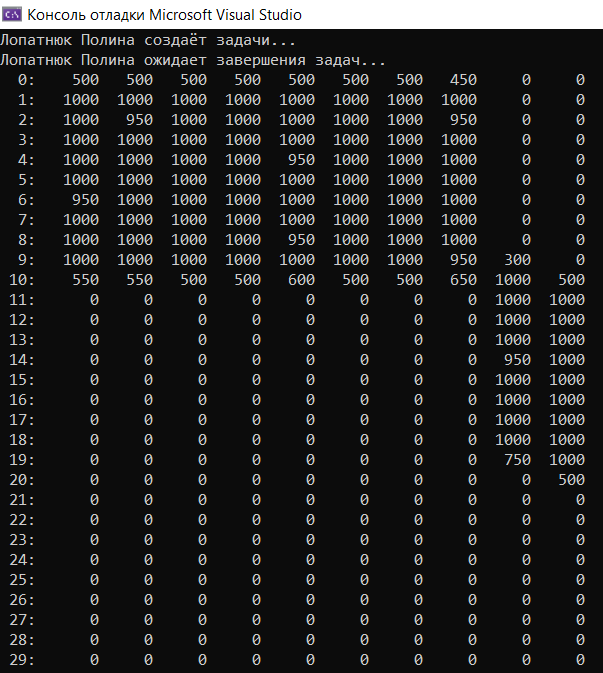
|  |
| --- |
| using System;  using System.Threading;  public class Program  {  const int ThreadCount = 10; // Количество потоков  const int ThreadLifeTime = 10; // Время работы каждого потока в секундах  const int ObservationTime = 30; // Время наблюдения в секундах  static int[,] Matrix = new int[ThreadCount, ObservationTime]; // Матрица для хранения данных о работе потоков  static DateTime StartTime = DateTime.Now;  static CountdownEvent countdown = new CountdownEvent(ThreadCount);  static void WorkThread(object o)  {  int id = (int)o;  for (int i = 0; i < ThreadLifeTime \* 20; i++)  {  DateTime CurrentTime = DateTime.Now;  int ElapsedSeconds = (int)Math.Round(CurrentTime.Subtract(StartTime).TotalSeconds);  if (ElapsedSeconds < ObservationTime)  {  Matrix[id, ElapsedSeconds] += 50;  }  MySleep(50);  }  countdown.Signal();  }  public static void MySleep(int milliseconds)  {  var startTime = DateTime.Now;  while (DateTime.Now - startTime < TimeSpan.FromMilliseconds(milliseconds))  {  int result = 1;  for (int i = 0; i < 10000; i++)  {  result \*= 2;  }  }  }  static void Main(string[] args)  {  Console.WriteLine("Полина Лопатнюк добавляет потоки в пул...");  for (int i = 0; i < ThreadCount; ++i)  {  object o = i;  ThreadPool.QueueUserWorkItem(WorkThread, o);  }  Console.WriteLine("Лопатнюк Полина ожидает завершения всех потоков...");  countdown.Wait();  for (int s = 0; s < ObservationTime; s++)  {  Console.Write("{0,3}: ", s);  for (int th = 0; th < ThreadCount; th++)  {  Console.Write(" {0,5}", Matrix[th, s]);  }  Console.WriteLine();  }  }  } |



**Задание 09.**

21. 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. Сравните результаты заданий 7-9 и запишите вывод в отчет.

|  |
| --- |
| public class Program  {  const int TaskCount = 10; // Количество задач  const int TaskLifeTime = 10; // Время жизни каждой задачи в секундах  const int ObservationTime = 30; // Время наблюдения в секундах  static int[,] Matrix = new int[TaskCount, ObservationTime]; // Матрица для хранения данных о работе задач  static DateTime StartTime = DateTime.Now;  static void Work(int id)  {  for (int i = 0; i < TaskLifeTime \* 20; i++)  {  DateTime CurrentTime = DateTime.Now;  int ElapsedSeconds = (int)Math.Round(CurrentTime.Subtract(StartTime).TotalSeconds);  if (ElapsedSeconds < ObservationTime)  {  Matrix[id, ElapsedSeconds] += 50;  }  MySleep(50);  }  }  public static void MySleep(int milliseconds)  {  var startTime = DateTime.Now;  while (DateTime.Now - startTime < TimeSpan.FromMilliseconds(milliseconds))  {  int result = 1;  for (int i = 0; i < 10000; i++)  {  result \*= 2;  }  }  }  static void Main(string[] args)  {  Task[] tasks = new Task[TaskCount];  Console.WriteLine("Лопатнюк Полина создаёт задачи...");  for (int i = 0; i < TaskCount; i++)  {  int taskId = i;  tasks[i] = Task.Run(() => Work(taskId));  }  Console.WriteLine("Лопатнюк Полина ожидает завершения задач...");  Task.WaitAll(tasks);  for (int s = 0; s < ObservationTime; s++)  {  Console.Write("{0,3}: ", s);  for (int th = 0; th < TaskCount; th++)  {  Console.Write(" {0,5}", Matrix[th, s]);  }  Console.WriteLine();  }  }  } |



**1. Использование класса Thread**

Этот метод предполагает явное создание потоков с помощью класса Thread. Каждый поток запускается отдельно и выполняет свои задачи параллельно.

Программа корректно выполняет задачи параллельно, но использование Thread при большом количестве задач может вызывать чрезмерное потребление ресурсов системы, что может снижать общую производительность.

**2. Использование пула потоков (ThreadPool)**

Пул потоков предоставляет систему управления потоками, которая эффективно использует ограниченное количество потоков для выполнения множества задач. Программа выполняется эффективно, потоков меньше, и система лучше управляет ресурсами. Однако наблюдаются небольшие задержки при большом количестве задач, так как система сама распределяет задачи между потоками пула.

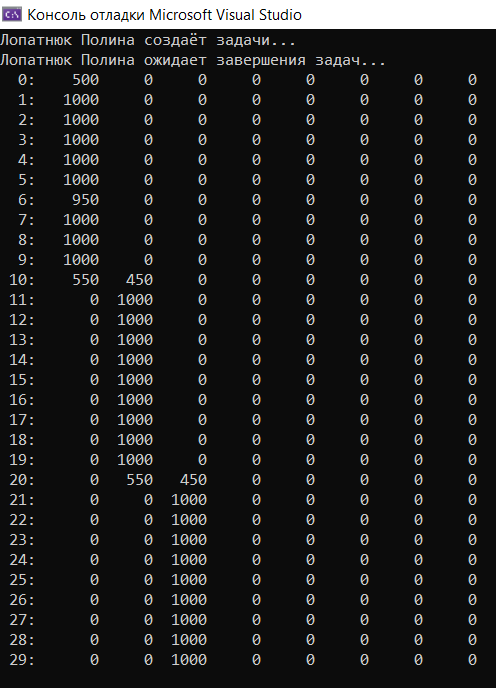
**3. Использование задач (Task)**

Задачи являются более современным подходом к многопоточности в C#. Они предоставляют высокоуровневую абстракцию и автоматическое управление потоками, что делает их идеальным выбором для большинства сценариев. Меньший контроль над низкоуровневыми потоками (передоверие системе управления потоками).

**Задание 10.**

23. 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.

|  |
| --- |
| public class Program  {  const int TaskCount = 8; // Уменьшено количество задач до 8  const int TaskLifeTime = 10; // Время жизни каждой задачи в секундах  const int ObservationTime = 30; // Время наблюдения в секундах  static int[,] Matrix = new int[TaskCount, ObservationTime]; // Матрица для хранения данных о работе задач  static DateTime StartTime = DateTime.Now;  static async Task Work(int id)  {  for (int i = 0; i < TaskLifeTime \* 20; i++)  {  DateTime CurrentTime = DateTime.Now;  int ElapsedSeconds = (int)Math.Round(CurrentTime.Subtract(StartTime).TotalSeconds);  if (ElapsedSeconds < ObservationTime)  {  Matrix[id, ElapsedSeconds] += 50;  }  await MySleep(50);  }  }  public static Task MySleep(int milliseconds)  {  return Task.Run(() =>  {  var startTime = DateTime.Now;  while (DateTime.Now - startTime < TimeSpan.FromMilliseconds(milliseconds))  {  int result = 1;  for (int i = 0; i < 10000; i++)  {  result \*= 2;  }  }  });  }  static async Task Main(string[] args)  {  Console.WriteLine("Лопатнюк Полина создаёт задачи...");  for (int i = 0; i < TaskCount; i++)  {  int taskId = i;  await Work(taskId);  }  Console.WriteLine("Лопатнюк Полина ожидает завершения задач...");  for (int s = 0; s < ObservationTime; s++)  {  Console.Write("{0,3}: ", s);  for (int th = 0; th < TaskCount; th++)  {  Console.Write(" {0,5}", Matrix[th, s]);  }  Console.WriteLine();  }  }  } |



**Задание 11**

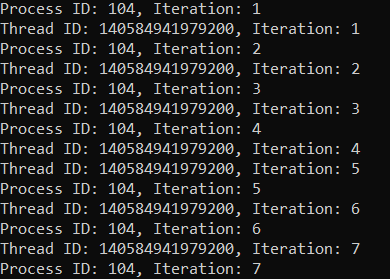
24. Разработайте на языке консольное Linux-приложение **OS04\_10** на языке С, выполняющее цикл 100 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса (использовать функции из pthread.h).

25. Процесс **OS04\_10** должен создать поток: потоковая функция **OS04\_10\_T1.**

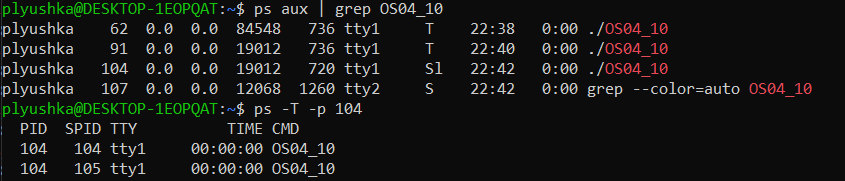
26. Поток **OS04\_10\_T1** - выполняет цикл 75 итераций с временной задержкой в 1 сек. с выводом на консоль идентификаторов процесса.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>  #include <unistd.h>  #define MAIN\_LOOP\_ITERATIONS 100  #define THREAD\_LOOP\_ITERATIONS 75  #define SLEEP\_TIME 1  void \*OS04\_10\_T1(void \*arg) {  for (int i = 0; i < THREAD\_LOOP\_ITERATIONS; i++) {  printf("Thread ID: %ld, Iteration: %d\n", pthread\_self(), i + 1);  sleep(SLEEP\_TIME);  }  return NULL;  }  int main() {  pthread\_t thread;  if (pthread\_create(&thread, NULL, OS04\_10\_T1, NULL) != 0) {  fprintf(stderr, "Error creating thread\n");  return 1;  }  for (int i = 0; i < MAIN\_LOOP\_ITERATIONS; i++) {  printf("Process ID: %d, Iteration: %d\n", getpid(), i + 1);  sleep(SLEEP\_TIME);  }  pthread\_join(thread, NULL);  return 0;  } |





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



**Задание 12. Ответьте на следующие вопросы**

1. **Что такое поток управления OS?**

Поток управления в операционной системе (ОС) — представляет собой наименьшую единицу обработки, которая может быть назначена ядром ОС. Потоки упрвления позволяют одной программе выполнять несколько последовательных операций параллельно.

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

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

);

Новый поток создается с помощью вызова функции **pthread\_create**. В качестве значения функции **возвращается идентификатор** только что созданного потока.

* pthread\_t \*thread: Указатель на переменную для хранения идентификатора потока.
* const pthread\_attr\_t \*attr: Указатель на атрибуты потока, 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)

**30. Что такое системные и пользовательские потоки?**

### Системные потоки

* **Определение**: Управляются операционной системой (ядром).
* **Управление**: Операционная система создает, планирует и завершает их выполнение.
* Независимы от процессов: Системные потоки не привязаны к конкретным процессам, они могут быть созданы и уничтожены независимо от процессов.
* **Применение**: Используются для выполнения низкоуровневых задач (системных задач).

### Пользовательские потоки

* **Определение**: Управляются приложением или библиотекой.
* **Управление**: Приложение или библиотека управляет их созданием и выполнением.
* Привязаны к процессам: Пользовательские потоки всегда связаны с определенным процессом.
* **Применение**: Используются для легковесных и высокопроизводительных задач.

Один системный поток может обслуживать несколько пользовательских потоков: Библиотека времени выполнения реализует многопоточность, используя один системный поток для выполнения нескольких пользовательских потоков.

Системные потоки обрабатывают операции ввода-вывода: Когда пользовательский поток выполняет операцию ввода-вывода, ядро Windows переключает его на системный поток, пока операция не будет завершена.

**31. Что такое многопоточность?**

**Многопоточность** — это концепция в программировании, позволяющая выполнять несколько задач (потоков выполнения) одновременно в рамках одного процесса. Потоки могут выполняться как одновременно, так и нет.

Одновременное выполнение потоков одного процесса называется **параллелизмом** (parallelism). Параллельное выполнение потоков в рамках одного процесса возможно только в многоядерных системах и не является обязательным поведением.

В одноядерных системах многопоточность может быть только **последовательной**. Переводя на программистский язык: многопоточный код не обязан быть по определению быстрым или параллельным, но быть таковым он может.

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

**Контекст потока** — это данные, которые операционная система сохраняет для того, чтобы поток мог быть приостановлен и затем возобновлен с того же места, где он был остановлен. Эти данные включают в себя состояние регистров процессора, указатель стека, счетчик команд и другую информацию, необходимую для корректного возобновления выполнения потока.

Контекст потока важен для того, чтобы операционная система могла переключаться между различными потоками, обеспечивая многозадачность. Когда поток приостанавливается (например, для переключения на другой поток), ОС сохраняет его контекст. Затем, при возобновлении выполнения потока, этот контекст восстанавливается, чтобы продолжить выполнение программы с того места, где оно было остановлено.

**33. 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.

**34. Что такое LWP?**

**LWP (Lightweight Process)**, или **легковесный процесс**, — это термин, который используется для обозначения поддерживаемых ядром пользовательских потоков. LWP — это промежуточная структура между пользовательскими потоками и потоками ядра. Она позволяет ядру операционной системы управлять параллельным выполнением нескольких потоков от имени процесса.

**35. Что такое потокобезопасность программного кода?**

**Потокобезопасность** — это предотвращение гонок данных (ситуация в многопоточной программе, когда два или более потока пытаются одновременно получить доступ к общей переменной (или ресурсу) и по крайней мере один из этих доступов является записью).

Процедура потокобезопасна, когда она логически корректна при одновременном выполнении несколькими потоками.

**36. Что такое реентерабельность кода?**

**Реентерабельность кода** — это свойство кода, позволяющее его безопасное выполнение несколькими потоками или процессами одновременно без риска ошибок или повреждения данных. Реентерабельный код можно повторно вызывать в любой момент, даже если предыдущий вызов не завершился, при этом каждый вызов будет корректно работать независимо от других.

**37. Что такое Fiber?**

**Fiber** (файбер, волокно) — это легковесная единица выполнения, которая управляется на уровне пользовательского пространства, а не операционной системой. В отличие от потоков, файберы не планируются операционной системой напрямую, а переключаются вручную с помощью программного кода. Это позволяет разработчику управлять переключением задач и распределением времени выполнения.