**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.

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

|  |  |
| --- | --- |
| КОД | #include <Windows.h>  #include <iostream>  using namespace std;  int main()  {  DWORD pid = GetCurrentProcessId();  DWORD tid = GetCurrentThreadId();  for (int i = 0; i < 1000; i++)  {  cout << i << " " << "PID: " << pid << " " << " TID: " << tid << " " << endl;  Sleep(1000);  }  } |
| Powershell |  |
| Performance  Monitor  (Монитор ресурсов) |  |

**Задание 02**

3. 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.

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

|  |  |
| --- | --- |
| Код | #include <Windows.h>  #include <iostream>  DWORD pid = NULL;  using namespace std;  DWORD WINAPI ChildThread() {  DWORD tid = GetCurrentThreadId();  for (int i = 0; i < 50; i++)  {  cout << i << " " << "OS04\_02\_T1: " << "PID: " << pid << " " << "TID: " << tid << ";" << endl;  Sleep(1000);  }  return 0;  }  DWORD WINAPI ChildSecondThread() {  DWORD tid = GetCurrentThreadId();  for (int i = 0; i < 125; i++)  {  cout << i << " " << "OS04\_02\_T2: " << "PID: " << pid << " " << "TID: " << tid << ";" << endl;  Sleep(1000);  }  return 0;  }  int main()  {  pid = GetCurrentProcessId();  DWORD tid = GetCurrentThreadId();  DWORD childId = NULL;  DWORD childSecondId = NULL;  // SECURITY\_ATTRIBUTES  HANDLE hChild = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ChildThread, NULL, 0, &childId);  HANDLE hChildSecond = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ChildSecondThread, NULL, 0, &childSecondId);  for (int i = 0; i < 100; i++)  {  cout << i << " " << "Parent Thread: " << "PID: " << pid << " " << "TID: " << tid << ";" << endl;  Sleep(1000);  }  WaitForSingleObject(hChild, INFINITE);  WaitForSingleObject(hChildSecond, INFINITE);  CloseHandle(hChild);  CloseHandle(hChildSecond);  } |
| Powershell |  |
| Performance  Monitor |  |

Из данных, представленных на графике, можно видеть, что в данный момент использование различных системных ресурсов находится на достаточно низком уровне. Например, использование процессора составляет около 10%, а свободная память близка к 100%.

**Задание 03.**

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

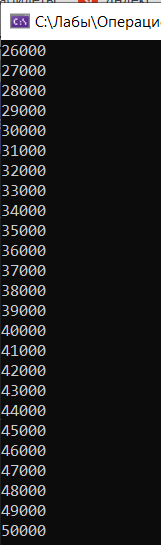
Сравните результаты между собой. От чего зависит максимальное количество потоков? Какое ограничение накладывает операционная система.

|  |  |
| --- | --- |
| Код | #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;  } |

 Программа в цикле создает 1 миллион потоков, используя функцию CreateThread(). Каждый поток выполняет функцию threadProc(), которая просто спит в течение 10 минут (600 секунд).

 Для каждой тысячной итерации цикла (каждые 1000 потоков) программа выводит текущий счетчик потоков в консоль.

 Вывод показывает, что программа успешно создает потоки до тех пор, пока не достигает определенного предела (в данном случае 16000 потоков), после чего перестает печатать новые значения.



Максимальное количество потоков, которые может создать приложение, зависит от комбинации факторов, включая ограничения операционной системы, архитектуру процессора, доступные ресурсы, настройки приложения и конфигурацию системы. Операционная система накладывает ограничения, чтобы предотвратить чрезмерное использование ресурсов и обеспечить стабильность работы.

#### Ограничения операционной системы:

* **Лимиты по ресурсам**: Операционная система накладывает лимиты на использование ресурсов, таких как память и дескрипторы. Если процесс пытается создать больше потоков, чем разрешено, или использует больше ресурсов, чем доступно, это приведет к ошибкам.
* **Потоковые дескрипторы**: Каждый поток использует дескриптор, и на систему может быть наложено ограничение на количество дескрипторов, которые могут быть открыты одновременно.

**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).

|  |  |
| --- | --- |
| Код | using System;  using System.Threading;  class Program  {  // Поток Z работает 10 секунд  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 ");  }  // Поток работает 20 секунд, параметр - строка-идентификатор  static void ThreadWithParam(object o)  {  for (int i = 0; i < 20; i++)  {  Console.Write(" ({0}-{1}) ", o.ToString(),  Thread.CurrentThread.ManagedThreadId);  Thread.Sleep(1000);  }  Console.WriteLine(" Поток {0} завершается ", o.ToString());  }  static void Main(string[] args)  {  var t1 = new Thread(ThreadZed);  var t1a = new Thread(ThreadWithParam);  var t1b = new Thread(ThreadWithParam);  t1.IsBackground = true;  t1a.IsBackground = true;  t1b.IsBackground = true;  t1.Start();  t1a.Start("Вероника"); // Имя  t1b.Start("Стрелковская"); // Фамилия  // Главный поток работает 5 секунд  for (int i = 0; i < 5; i++)  {  Console.Write(" (\*-{0}) ", Thread.CurrentThread.ManagedThreadId);  Thread.Sleep(1000);  }  Console.WriteLine("Главный поток завершается");  }  } |
| 10 | На первом этапе приложение завершится через 5 секунд, так как главный поток завершает выполнение, и фоновые потоки будут остановлены |
| 11 | После изменения значения t1.IsBackground = false;  На втором этапе, когда первый поток не является фоновым, приложение будет работать до тех пор, пока этот поток не завершится, то есть около 10 секунд. |
| 12 | После изменения значения t2.IsBackground = false;  На третьем этапе, когда второй поток не является фоновым, приложение будет работать до завершения этого потока, что составит около 20 секунд. |

 В Main() создаются три потока:

* Поток t1 - выполняет функцию ThreadZed(), которая работает 10 секунд.
* Поток t1a - выполняет функцию ThreadWithParam("Вероника"), которая работает 20 секунд.
* Поток t1b - выполняет функцию ThreadWithParam("Стрелковская"), которая также работает 20 секунд.

 Все три дочерних потока помечены как фоновые (IsBackground = true), что означает, что они будут автоматически завершены, когда завершится главный поток.

 Главный поток выполняет цикл в течение 5 секунд, выводя идентификатор своего потока (\*-1).

 В результате вывода:

* Каждую секунду выводится информация о работе всех четырех потоков: главный поток, два дочерних потока с именами и поток Z.
* Через 10 секунд поток Z завершается.
* Через 20 секунд завершаются два дочерних потока с именами.
* Наконец, главный поток завершается.

**Фоновые потоки:** Когда поток является фоновым, он завершает свою работу, как только завершается главный поток. Это объясняет, почему приложение завершилось через 5 секунд в первом случае.

**Нефоновые потоки:** Потоки, не являющиеся фоновыми, блокируют завершение приложения, пока они не завершатся. Это объясняет, почему время выполнения увеличивается до 10 или 20 секунд в случае изменения свойств IsBackground.

**Задание 05.**

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;  using System.Diagnostics;  class Program  {  static void Main(string[] args)  {  // Тестируем MySleep на 10 секунд  int duration = 10000; // 10 секунд в миллисекундах  Console.WriteLine($"Запуск MySleep на {duration} миллисекунд...");  // Запускаем таймер  Stopwatch stopwatch = Stopwatch.StartNew();  double result = MySleep(duration);  stopwatch.Stop();  // Выводим результат  Console.WriteLine($"MySleep завершился за {stopwatch.ElapsedMilliseconds} миллисекунд.");  Console.WriteLine($"Результат вычислений: {result}");  }  static double MySleep(int ms)  {  double sum = 0;  DateTime startTime = DateTime.Now;  while ((DateTime.Now - startTime).TotalMilliseconds < ms)  {  double temp = 0.711 + (DateTime.Now - startTime).TotalSeconds / 10.0;  double a, b, c, d, e, nt;  for (int k = 0; k < 5500; ++k)  {  nt = temp - k / 27000.0;  a = Math.Sin(nt);  b = Math.Cos(nt);  c = Math.Cos(nt / 2.0);  d = Math.Sin(nt / 2);  e = Math.Abs(1.0 - a \* a - b \* b) + Math.Abs(1.0 - c \* c - d \* d);  sum += e;  }  Thread.Sleep(1); // Спим 1 миллисекунду, чтобы не перегружать ЦП  }  return sum;  }  } |
|  |

**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.

|  |
| --- |
| using System;  using System.Threading;  class Program  {  // Общая переменная  static int Count = 0;  static void WorkThread()  {  for (int i = 0; i < 5000000; ++i)  {  // Увеличиваем счетчик без блокировки  Count++;  }  }  static void Main(string[] args)  {  Thread[] threads = new Thread[20];  // Запускаем 20 потоков  for (int i = 0; i < 20; ++i)  {  threads[i] = new Thread(WorkThread);  threads[i].Start();  }  // Ожидаем завершения всех потоков  for (int i = 0; i < 20; ++i)  threads[i].Join(); // Метод Join используется для ожидания завершения каждого потока  // Выводим результат  Console.WriteLine("Общее значение Count: " + Count);  Console.WriteLine("Ожидаемое значение: " + (20 \* 5000000));  }  } |
|  |

Результат, выведенный в консоли, демонстрирует проблему гонки данных (race condition) в многопоточном приложении. Несмотря на то, что каждый из 20 потоков увеличивает значение переменной Count 5 000 000 раз, итоговое значение Count не равно ожидаемому значению 20 \* 5000000 = 100 000 000.

Это происходит из-за того, что в многопоточном приложении несколько потоков могут одновременно обращаться к общей переменной Count и изменять ее значение. Поскольку операция увеличения Count++ не является атомарной, возникает гонка данных, и некоторые из приращений могут быть "потеряны".

**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)

|  |
| --- |
| using System;  using System.Threading;  class Program  {  const int ThreadCount = 16; // Количество потоков  const int ThreadLifeTime = 10; // Продолжительность работы каждого потока в секундах  const int ObservationTime = 30; // Время наблюдения в секундах  static int[,] Matrix = new int[ThreadCount, ObservationTime];  static DateTime StartTime = DateTime.Now;  // Метод, который выполняет вычисления  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 - 0.49);  if (ElapsedSeconds >= 0 && ElapsedSeconds < ObservationTime) // Проверка на границы  {  Matrix[id, ElapsedSeconds] += 50;  }  MySleep(50); // Вычисления, которые занимают время  }  }  // Метод для выполнения "сна" в течение заданного времени  static Double MySleep(int ms)  {  double sum = 0, temp;  for (int t = 0; t < ms; ++t)  {  temp = 0.711 + (double)t / 10000.0;  for (int k = 0; k < 5500; ++k)  {  double nt = temp - k / 27000.0;  double a = Math.Sin(nt);  double b = Math.Cos(nt);  double c = Math.Cos(nt / 2.0);  double d = Math.Sin(nt / 2);  double e = Math.Abs(1.0 - a \* a - b \* b) + Math.Abs(1.0 - c \* c - d \* d);  sum += e;  }  }  return sum;  }  static void Main(string[] args)  {  Thread[] t = new Thread[ThreadCount];  for (int i = 0; i < ThreadCount; ++i)  {  object o = i;  t[i] = new Thread(WorkThread);  t[i].Start(o);  }  Console.WriteLine("Ожидание завершения потоков...");  for (int i = 0; i < ThreadCount; ++i)  {  t[i].Join();  }  // Вывод результатов в виде таблицы  Console.WriteLine("Результаты работы потоков:");  Console.WriteLine("Секунда | " + string.Join(" | ", Array.ConvertAll(new int[ThreadCount], x => $"Поток {Array.IndexOf(new int[ThreadCount], x)}")));  Console.WriteLine(new string('-', 60));  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.**

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;  class Program  {  const int ThreadCount = 16; // Количество потоков  const int ThreadLifeTime = 10; // Продолжительность работы каждого потока в секундах  const int ObservationTime = 30; // Время наблюдения в секундах  static int[,] Matrix = new int[ThreadCount, ObservationTime];  static DateTime StartTime = DateTime.Now;  // Метод, который выполняет вычисления  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 - 0.49);  // Проверка на границы  if (ElapsedSeconds >= 0 && ElapsedSeconds < ObservationTime)  {  Matrix[id, ElapsedSeconds] += 50;  }    MySleep(50); // Вычисления, которые занимают время  }  }  // Метод для выполнения "сна" в течение заданного времени  static Double MySleep(int ms)  {  double sum = 0, temp;  for (int t = 0; t < ms; ++t)  {  temp = 0.711 + (double)t / 10000.0;  for (int k = 0; k < 5500; ++k)  {  double nt = temp - k / 27000.0;  double a = Math.Sin(nt);  double b = Math.Cos(nt);  double c = Math.Cos(nt / 2.0);  double d = Math.Sin(nt / 2);  double e = Math.Abs(1.0 - a \* a - b \* b) + Math.Abs(1.0 - c \* c - d \* d);  sum += e;  }  }  return sum;  }  static void Main(string[] args)  {  Console.WriteLine("A student ... is placing threads to the pool...");  // Запускаем потоки в пуле  for (int i = 0; i < ThreadCount; ++i)  {  object o = i;  ThreadPool.QueueUserWorkItem(WorkThread, o);  }  Console.WriteLine("A student ... is waiting for the threads to finish...");  Thread.Sleep(1000 \* ObservationTime);  // Вывод результатов в виде таблицы  Console.WriteLine("Результаты работы потоков:");  Console.WriteLine("Секунда | " + string.Join(" | ", Array.ConvertAll(new int[ThreadCount], x => $"Поток {Array.IndexOf(new int[ThreadCount], x)}")));  Console.WriteLine(new string('-', 60));  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.**

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

|  |
| --- |
| using System;  using System.Threading.Tasks;  class Program  {  const int TaskCount = 16; // Количество задач  const int TaskLifeTime = 10; // Продолжительность работы каждой задачи в секундах  const int ObservationTime = 30; // Время наблюдения в секундах  static int[,] Matrix = new int[TaskCount, ObservationTime];  static DateTime StartTime = DateTime.Now;  // Метод, который выполняет вычисления  static void Work(object o)  {  int id = (int)o;  for (int i = 0; i < TaskLifeTime \* 20; i++)  {  DateTime CurrentTime = DateTime.Now;  int ElapsedSeconds = (int)Math.Round(CurrentTime.Subtract(StartTime).TotalSeconds - 0.49);  // Проверка на границы  if (ElapsedSeconds >= 0 && ElapsedSeconds < ObservationTime)  {  Matrix[id, ElapsedSeconds] += 50;  }  MySleep(50); // Вычисления, которые занимают время  }  }  // Метод для выполнения "сна" в течение заданного времени  static Double MySleep(int ms)  {  double sum = 0, temp;  for (int t = 0; t < ms; ++t)  {  temp = 0.711 + (double)t / 10000.0;  for (int k = 0; k < 5500; ++k)  {  double nt = temp - k / 27000.0;  double a = Math.Sin(nt);  double b = Math.Cos(nt);  double c = Math.Cos(nt / 2.0);  double d = Math.Sin(nt / 2);  double e = Math.Abs(1.0 - a \* a - b \* b) + Math.Abs(1.0 - c \* c - d \* d);  sum += e;  }  }  return sum;  }  static void Main(string[] args)  {  Task[] tasks = new Task[TaskCount];  int[] numbers = new int[TaskCount];  for (int i = 0; i < TaskCount; i++)  numbers[i] = i;  Console.WriteLine("A student ... is creating tasks...");  // Создание задач  for (int i = 0; i < TaskCount; i++)  {  int taskId = i; // Локальная переменная для избежания замыкания  tasks[i] = Task.Run(() => Work(taskId));  }  Console.WriteLine("A student ... is waiting for tasks to finish...");  Task.WaitAll(tasks);  // Вывод результатов в виде таблицы  Console.WriteLine("Результаты работы задач:");  Console.WriteLine("Секунда | " + string.Join(" | ", Array.ConvertAll(new int[TaskCount], x => $"Задача {Array.IndexOf(new int[TaskCount], x)}")));  Console.WriteLine(new string('-', 60));  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. **Сравнение методов**:
   * **Задание 7 (Потоки)**: Каждый поток создавался вручную, что могло привести к накладным расходам на управление потоками.
   * **Задание 8 (Пул потоков)**: Использование пула потоков позволяет эффективно управлять потоками, перерабатывая их по мере необходимости и минимизируя накладные расходы.
   * **Задание 9 (Задачи)**: Использование Task предоставляет еще более высокоуровневый подход для работы с асинхронным программированием, упрощая управление и обработку результатов.
2. **Производительность**:
   * В зависимости от конфигурации системы и нагрузки на процессоры, производительность может варьироваться. Задачи могут быть более эффективными в использовании ресурсов по сравнению с ручным управлением потоками.

**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.

|  |
| --- |
| using System;  using System.Threading.Tasks;  class Program  {  const int TaskCount = 4; // Количество задач, равное количеству логических процессоров  const int TaskLifeTime = 10; // Продолжительность работы каждой задачи в секундах  const int ObservationTime = 30; // Время наблюдения в секундах  static int[,] Matrix = new int[TaskCount, ObservationTime];  static DateTime StartTime = DateTime.Now;  // Метод, который выполняет вычисления  static void Work(object o)  {  int id = (int)o;  for (int i = 0; i < TaskLifeTime \* 20; i++)  {  DateTime CurrentTime = DateTime.Now;  int ElapsedSeconds = (int)Math.Round(CurrentTime.Subtract(StartTime).TotalSeconds - 0.49);  // Проверка на границы  if (ElapsedSeconds >= 0 && ElapsedSeconds < ObservationTime)  {  Matrix[id, ElapsedSeconds] += 50;  }  MySleep(50); // Вычисления, которые занимают время  }  }  // Метод для выполнения "сна" в течение заданного времени  static Double MySleep(int ms)  {  double sum = 0, temp;  for (int t = 0; t < ms; ++t)  {  temp = 0.711 + (double)t / 10000.0;  for (int k = 0; k < 5500; ++k)  {  double nt = temp - k / 27000.0;  double a = Math.Sin(nt);  double b = Math.Cos(nt);  double c = Math.Cos(nt / 2.0);  double d = Math.Sin(nt / 2);  double e = Math.Abs(1.0 - a \* a - b \* b) + Math.Abs(1.0 - c \* c - d \* d);  sum += e;  }  }  return sum;  }  static void Main(string[] args)  {  Task[] tasks = new Task[TaskCount];  Console.WriteLine("A student ... is creating tasks...");  // Создание задач с последовательным выполнением  for (int i = 0; i < TaskCount; i++)  {  int taskId = i; // Локальная переменная для избежания замыкания  if (i == 0)  {  tasks[i] = Task.Run(() => Work(taskId));  }  else  {  tasks[i] = tasks[i - 1].ContinueWith(t => Work(taskId));  }  }  Console.WriteLine("A student ... is waiting for tasks to finish...");  Task.WaitAll(tasks);  // Вывод результатов в виде таблицы  Console.WriteLine("Результаты работы задач:");  Console.WriteLine("Секунда | " + string.Join(" | ", Array.ConvertAll(new int[TaskCount], x => $"Задача {Array.IndexOf(new int[TaskCount], x)}")));  Console.WriteLine(new string('-', 60));  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.**

1) 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).

2) Процесс OS04\_10 должен создать поток: потоковая функция OS04\_10\_T1.

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

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

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

~~В 10 изменить цикл~~

~~7—9 – количество 16~~

~~1-2 через программку perfmon~~

~~Без lock в 6~~

8 и 9 почему так выводит. Как работает пул и таск

using System;

using System.Threading;

class Program

{

// Общая переменная

static int Count = 0;

// Объект для синхронизации

static readonly object \_lock = new object();

static void WorkThread()

{

for (int i = 0; i < 5000000; ++i)

{

// Используем блокировку для обеспечения безопасного доступа к переменной

//Она гарантирует, что только один поток может выполнять код внутри блока lock в любой момент времени.

//Это предотвращает одновременное изменение переменной Count несколькими потоками, что могло бы привести к некорректному результату.

lock (\_lock)

{

Count++;

}

}

}

static void Main(string[] args)

{

Thread[] threads = new Thread[20];

// Запускаем 20 потоков

for (int i = 0; i < 20; ++i)

{

threads[i] = new Thread(WorkThread);

threads[i].Start();

}

// Ожидаем завершения всех потоков

for (int i = 0; i < 20; ++i)

threads[i].Join();//Метод Join используется для ожидания завершения каждого потока

// Выводим результат

Console.WriteLine("Общее значение Count: " + Count);

Console.WriteLine("Ожидаемое значение: " + (20 \* 5000000));

}

}

ОТВЕТЫ НА ВОПРОСЫ

# Что такое поток управления OS?

**Поток (управления) OS** –последовательность инструкций, выполняемых процессором в выделенные OS интервалы времени

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

CreateThread(Windows)

Pthread\_create(linux)

# Что такое системные и пользовательские потоки?

**Системный поток:** выполняют различные сервисы ОС, запускаются ядром ОС.

**Пользовательские потоки:** служат для решения пользовательских задач, запускаются приложением

# Что такое многопоточность?

**Многопоточность** – способность программы выполнять несколько потоков одновременно. **Приоритетная и кооперативная** многопоточность.

# Что такое контекст потока и для чего он нужен?

**Контекст потока** – набор данных, который определяет состояние потока в любой момент времени. Включает в себя:

* программный код;
* набор регистров;
* стек памяти;
* стек ядра ОС;
* маркер доступа.

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

* **Выполняемый –** выполняется на процессоре в данный момент;
* **Заблокированный –** приостановлен, ожидает завершения другого потока;
* **Готовый –** готов к выполнению и ожидает, когда ему будет предоставлено время процессора;
* **Завершенный** **–** завершил выполнение и освободил все ресурсы.

# Что такое LWP?

**Легковесный процесс** – термин, используемый для обозначения потока, который является независимым элементом выполнения в рамках процесса. LWP имеет свой соб ственный контекст выполнения, включая регистры и стек, но делит память и ресурсы с другими потоками в рамках того же процесса.

# Что такое потокобезопасность программного кода?

**Потокобезопасность кода –** свойство кода корректно работать в нескольких потоках одновременно.

# Реентерабельность кода – свойство одной копии кода работать в нескольких потоках одновременно.

# Что такое Fiber?

Легковесный поток, который управляется приложением, а не ОС. Fibers позволяют переключаться между задачами внутри одного потока, что может быть полезным для управления задачами в высоконагруженных системах.