Задание 01

1. Разработайте консольное Windows-приложение OS05\_01 на языке С++, выводящее на консоль следующую информации:

* идентификатор текущего процесса;
* идентификатор текущего (main) потока;
* приоритет (приоритетный класс) текущего процесса;
* приоритет текущего потока;
* маску (affinity mask) доступных процессу процессоров в двоичном виде;
* количество процессоров, доступных процессу;
* процессор, назначенный текущему потоку.

|  |
| --- |
| #include <Windows.h>  #include <iostream>  using namespace std;  //функция приоритета процесса  void printProcessPrty(HANDLE h)  {  DWORD prty = GetPriorityClass(h);//приоритетный класс текущего процесса  std::cout << " --- Current PID = " << GetCurrentProcessId() << " \n";  switch (prty)//для определения класса приоритета  {  case IDLE\_PRIORITY\_CLASS:  cout << " ----+ Priority of process = IDLE\_PRIORITY\_CLASS \n";  break;  case BELOW\_NORMAL\_PRIORITY\_CLASS:  cout << " ----+ Priority of process = BELOW\_NORMAL\_PRIORITY\_CLASS \n";  break;  case NORMAL\_PRIORITY\_CLASS:  cout << " ----+ Priority of process = NORMAL\_PRIORITY\_CLASS \n";  break;  case ABOVE\_NORMAL\_PRIORITY\_CLASS:  cout << " ----+ Priority of process = ABOVE\_NORMAL\_PRIORITY\_CLASS \n";  break;  case HIGH\_PRIORITY\_CLASS:  cout << " ----+ Priority of process = HIGH\_PRIORITY\_CLASS \n";  break;  case REALTIME\_PRIORITY\_CLASS:  cout << " ----+ Priority of process = REALTIME\_PRIORITY\_CLASS \n";  break;  default:  cout << " ----+ Priority of process = ? \n";  break;  }  return;  }  //функция приоритета потока  void printThreadPrty(HANDLE h)  {  DWORD icpu = SetThreadIdealProcessor(h, MAXIMUM\_PROCESSORS);//Устанавливает идеальный процессор для потока. MAXIMUM\_PROCESSORS указывает максимальное количество процессоров, доступных системе  DWORD prty = GetThreadPriority(h);//приоритет потока  std::cout << " --- Current Thread ID = " << GetCurrentThreadId() << "\n";  std::cout << " ----+ priority = " << prty << " \n";  switch (prty)  {  case THREAD\_PRIORITY\_LOWEST:  cout << " ----+ Priority of thread = THREAD\_PRIORITY\_LOWEST \n";  break;  case THREAD\_PRIORITY\_BELOW\_NORMAL:  cout << " ----+ Priority of thread = THREAD\_PRIORITY\_BELOW\_NORMAL \n";  break;  case THREAD\_PRIORITY\_NORMAL:  cout << " ----+ Priority of thread = THREAD\_PRIORITY\_NORMAL \n";  break;  case THREAD\_PRIORITY\_ABOVE\_NORMAL:  cout << " ----+ Priority of thread = THREAD\_PRIORITY\_ABOVE\_NORMAL \n";  break;  case THREAD\_PRIORITY\_HIGHEST:  cout << " ----+ Priority of thread = THREAD\_PRIORITY\_HIGHEST \n";  break;  case THREAD\_PRIORITY\_IDLE:  cout << " ----+ Priority of thread = THREAD\_PRIORITY\_IDLE \n";  break;  case THREAD\_PRIORITY\_TIME\_CRITICAL:  cout << " ----+ Priority of thread = THREAD\_PRIORITY\_TIME\_CRITICAL \n";  break;  default:  cout << " ----+ Priority of thread = ? \n";  break;  }  cout << " ----+ IdealProcessor = " << icpu << " \n"; //номер процессора, на который установлен идеальный процессор для потока  return;  }  string toBinary(int n)  {  std::string r;  while (n != 0) { r = (n % 2 == 0 ? "0" : "1") + r; n /= 2; }  return r;  }  // Функция для подсчета количества доступных процессоров  int countAvailableProcessors(DWORD\_PTR affinityMask)  {  int count = 0;  while (affinityMask)  {  count += (affinityMask & 1); // Увеличиваем счетчик, если младший бит равен 1  affinityMask >>= 1; // Сдвигаем маску вправо  }  return count;  }  int main()  {  HANDLE hp = GetCurrentProcess();  HANDLE ht = GetCurrentThread();  printProcessPrty(hp);  printThreadPrty(ht);  try  {  {  DWORD\_PTR pa = NULL, sa = NULL, icpu = -1;//для масок процессоров  if (!GetProcessAffinityMask(hp, &pa, &sa))//Получает маску процессоров, доступных для текущего процесса  throw "GetProcessAffinityMask";  cout << " Process Affinity Mask: " << toBinary(pa) << "\n";  cout << " System Affinity Mask: " << toBinary(sa) << "\n";  // Подсчет количества доступных процессоров для текущего процесса  int availableProcessors = countAvailableProcessors(pa);  cout << " ----+ Available Processors for current process = " << availableProcessors << " \n";  }  }  catch (char\* msg)  {  cout << "Error " << msg << "\n";  }  system("pause");  return 0;  } |
|  |

Задание 02

2. Создайте консольное Windows OS05\_02 на языке С#, взяв за основу приложение OS04\_07 из Лабораторной работы №4. Измените метод Main таким образом, чтобы потоки 0, 3, 6 и т.д. запускались с минимальным приоритетом потока, а потоки 2, 5, 8... – с максимальным. Класс приоритета процесса оставьте по умолчанию (Normal). Как пример,

можно использовать следующий фрагмент кода.

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

{

object o = i;

t[i] = new Thread(WorkThread);

switch (i % 3)

{

case 0:

t[i].Priority = ThreadPriority.Lowest;

break;

case 2:

t[i].Priority = ThreadPriority.Highest;

break;

}

t[i].Start(o);

}

|  |
| --- |
| 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];//хранение результатов работы каждого потока в течение времени наблюдения  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;  //Вычисляет количество секунд, прошедших с момента старта, с небольшим сдвигом (уменьшение на 0.49 секунды для округления)  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);  // Установка приоритета потоков  switch (i % 3)  {  case 0:  t[i].Priority = ThreadPriority.Lowest;  break;  case 2:  t[i].Priority = ThreadPriority.Highest;  break;  }  t[i].Start(o);  }  Console.WriteLine("Ожидание завершения потоков...");  for (int i = 0; i < ThreadCount; ++i)  {  t[i].Join();//чтобы главный поток ожидал завершения всех созданных потоков  }  // Вывод результатов в виде таблицы  Console.WriteLine("Результаты работы потоков:");  //Создает новый массив длиной ThreadCount (в вашем случае 10) и заполняет его строками вида "Поток 0", "Поток 1" и т.д.  //Объединяет все строки из массива в одну строку, разделяя их символами " | "  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();  }  }  }  } |
| если поток не успевает обновить массив в определенный временной интервал, то он просто не вносит изменения |

Задание 03

5. Создайте консольное Windows OS05\_03 на языке С#, взяв за основу приложение OS05\_02 из настоящей работы. На этот раз только несколько потоков запустите на

наименьшем приоритете потока, а остальные – на наибольшем.

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

{

object o = i;

t[i] = new Thread(WorkThread);

if (i < 2) // здесь 2 - половина логических процессоров

t[i].Priority = ThreadPriority.Lowest;

else

t[i].Priority = ThreadPriority.Highest;

t[i].Start(o);

}

6. Выполните приложение OS05\_03 с другими парами приоритетов, например, BelowNormal и Normal. Изменился ли характер работы потоков?.

* **Lowest**: Минимальный приоритет. Потоки с этим приоритетом будут исполняться только в том случае, если нет других потоков с более высоким приоритетом.
* **BelowNormal**: Приоритет ниже нормального. Используется для потоков, которые не требуют быстрого выполнения и могут подождать, пока более важные задачи будут завершены.
* **Normal**: Нормальный приоритет. Это стандартный уровень приоритета для потоков. Большинство потоков, если не указано иное, получают этот уровень.
* **AboveNormal**: Приоритет выше нормального. Потоки с этим приоритетом будут выполняться быстрее, чем потоки с нормальным приоритетом, но медленнее, чем потоки с высоким приоритетом.
* **Highest**: Максимальный приоритет. Потоки с этим приоритетом получают максимальное время процессора. Используется для критически важных задач, которые должны выполняться как можно быстрее.

|  |
| --- |
| 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];//хранение результатов работы каждого потока в течение времени наблюдения  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;  //Вычисляет количество секунд, прошедших с момента старта, с небольшим сдвигом (уменьшение на 0.49 секунды для округления)  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);  // Установка приоритета потоков  if (i < 2) // здесь 2 - половина логических процессоров  t[i].Priority = ThreadPriority.Lowest;  else  t[i].Priority = ThreadPriority.Highest;  t[i].Start(o);  }  Console.WriteLine("Ожидание завершения потоков...");  for (int i = 0; i < ThreadCount; ++i)  {  t[i].Join();//чтобы главный поток ожидал завершения всех созданных потоков  }  // Вывод результатов в виде таблицы  Console.WriteLine("Результаты работы потоков:");  //Создает новый массив длиной ThreadCount (в вашем случае 10) и заполняет его строками вида "Поток 0", "Поток 1" и т.д.  //Объединяет все строки из массива в одну строку, разделяя их символами " | "  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 04

9. Разработайте консольное Linux-приложение OS05\_04 на языке С++, выводящее на консоль следующую информации:

* идентификатор текущего процесса;
* идентификатор текущего (main) потока;
* приоритет (nice) текущего потока;
* номера доступных процессоров.

|  |
| --- |
| #include <stdio.h>  #include <pthread.h>  #include <sched.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/syscall.h>  pid\_t gettid() {  return (pid\_t)syscall(SYS\_gettid);  }  int main() {  // Идентификатор текущего процесса  pid\_t pid = getpid();  printf("Идентификатор текущего процесса: %d\n", pid);  // Идентификатор текущего (main) потока  pid\_t tid = gettid();  printf("Идентификатор текущего потока: %d\n", tid);  // Приоритет (nice) текущего потока  int nice\_value = nice(0); //Функция nice позволяет задать приоритет выполнения, где меньшие значения соответствуют более высокому приоритету. В данном случае 0 означает, что приоритет не изменяется.  printf("Приоритет (nice) текущего потока: %d\n", nice\_value);  // Номера доступных процессоров  int num\_cpus = sysconf(\_SC\_NPROCESSORS\_ONLN);  printf("Количество доступных процессоров: %d\n", num\_cpus);  return 0;  } |
|  |

Задание 5

Разработайте консольное Linux-приложение OS05\_05 на языке С, выполняющее длинный цикл.

Запустите приложение OS05\_05. Зафиксируйте < > текущее значение nicе, полученное с

помощью команды top.

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  int main() {  // Бесконечный цикл  while (1) {  // Можно добавить задержку для уменьшения нагрузки на процессор  usleep(100000); // Задержка 100 мс  }  return 0;  } |
|  **PID**: Идентификатор процесса (Process ID).   **USER**: Имя пользователя, запустившего процесс.   **PR**: Приоритет процесса.   **NI**: "Nice" значение процесса, определяющее приоритет.   **VIRT**: Общее количество виртуальной памяти, используемой процессом.   **RES**: Количество физической памяти, используемой процессом (в реальной RAM).   **SHR**: Количество разделяемой памяти, используемой процессом.   **S**: Состояние процесса (например, R - выполняется, S - спит, Z - зомби и т.д.).   **%CPU**: Процент использования CPU процессом.   **%MEM**: Процент использования физической памяти процессом.   **TIME+**: Время, в течение которого процесс использовал CPU.   **COMMAND**: Команда или имя процесса. |
| #include <stdio.h> #include <unistd.h> #include <stdlib.h> #include <sys/resource.h>  int main() {  if (nice(19) == -1) { perror("Priority error"); exit(EXIT\_FAILURE);} // Бесконечный цикл while (1) { // Можно добавить задержку для уменьшения нагрузки на процессор usleep(100000); // Задержка 100 мс } return 0;  } |
| С -20 не получается ? |

Чтобы изменить значение приоритета мне нужны определённые права пользователя (использование команды sudo возможно только администратору, и я не могу выдать себе эти права. Не знаю).

Вообще для дальнейшего выполнения мы должны использовать команду:

sudo renice -20 4897 для увеличения приоритета до максимального

sudo renice 19 4897 для уменьшения приоритета до минимального

**Ответы на вопросы**

1. **Поясните понятие «мультизадачная OS с вытеснением».**

ОС, которая поддерживает одновременное выполнение нескольких задач (процессов) и может приостанавливать выполнение одной задачи в пользу другой.

1. **Поясните понятие «циклическое планирование».**

Метод планирования, при котором каждый процесс получает фиксированный квант времени на выполнение, после чего, если не завершен, переходит в конец очереди.

1. **Поясните понятие «приоритетное планирование».**

Подход, при котором каждому процессу присваивается приоритет, и процессор отдается процессу с наивысшим приоритетом.

1. **Поясните понятие «кооперативное планирование».**

Метод, где процессы сами решают, когда освободить процессор, без вынуждения со стороны планировщика.

1. **Поясните понятие «OS реального времени».**

ОС, обеспечивающая выполнение задач в строго заданных временных рамках для обработки данных в реальном времени.

1. **Поясните понятие «приоритет процесса».**

Значение, определяющее важность процесса относительно других, используемое при приоритетном планировании.

1. **Поясните выражение «поток уступает процессор другому потоку».**

Поток завершает свое выполнение или переходит в ожидание, предоставляя процессор другому потоку.

1. **Windows: как поток может уступить процессор?**

Поток может вызвать функцию Sleep или SwitchToThread.

1. **Windows: что такое базовый приоритет потока, как он вычисляется и диапазон его изменения?**

Значение, влияющее на приоритет потока, вычисляемое на основе приоритета процесса и планировщика. Базовый приоритет потока может принимать значения в диапазоне от 1 до 31.

1. **Windows: поясните назначение и принцип применения системного вызова ResumeThread.**

Системный вызов для возобновления выполнения приостановленного потока.

1. **Windows: поясните назначение и принцип применения системного вызова WaitForSingleObject.**

Системный вызов для ожидания завершения одного объекта, например, потока или процесса.

1. **Windows: поясните назначение и принцип применения системных вызовов GetProcessPriorityBoost, GetThreadPriorityBoost, SetProcessPriorityBoost, SetThreadPriorityBoost.**

Системные вызовы для управления приоритетом ускорения процесса или потока.

1. **Linux: поясните принцип идентификации процессов и потоков и поясните почему он такой.**

Процессы и потоки в Linux идентифицируются уникальными PID (Process ID) и TID (Thread ID) соответственно.

1. **Linux: Поясните понятие «планировщик потоков».**

Механизм планирования, который управляет выполнением потоков.

1. **Linux: поясните принцип использования значения nice –процесса, диапазон его изменения, для какого режима работы планировщика это значение применяется?**

Значение nice - это целое число, которое определяет приоритет процесса. Чем ниже значение nice, тем выше приоритет процесса. Диапазон от -20(max) до 19(min). Значение nice применяется в контексте вытесняющей многозадачности.

1. **Linux: перечислите политики планирования, какая действует по умолчанию?**

SCHED\_OTHER (по умолчанию) - политика для обычных процессов

SCHED\_FIFO - политика с жесткими временными ограничениями

SCHED\_RR (карусельная (round-robin)) - политика с жесткими временными ограничениями и квази-раунд-робин планированием

SCHED\_BATCH - политика для процессов, выполняющих задачи с низким приоритетом

SCHED\_IDLE - политика с наименьшим приоритетом

1. **Linux: с помощью какого системного вызова поток может уступить процессор.**

Системный вызов sched\_yield.