Лабораторная работа №1

По Архитектурам вычислительных систем

Выполнил: Лебедев В.В., 18-ПРИ

**Задание 1 (на паре)**

**Задание 2.** Разработать программу, содержащую один параллельный регион. До входа в параллельную область на экран выводится текст «Последовательная область 1», в параллельной области каждая нить печатает текст «Параллельная область», после завершения параллельной области оставшаяся нить-мастер печатает текст «Последовательная область 2».

**Задание 3.** Разработать параллельную программу, нить-мастер которой печатает в консоли общее количество порождённых нитей, а остальные нити печатают свой порядковый номер.

**Задание 4.** Разработать программу, состоящую из трех параллельных регионов. В каждом расположена функция вывода строки на консоль вида «Параллельный регион N», где N – номер региона. Для первого региона установить для исполнения две нити с помощью переменной окружения в настройках проекта, для второго – три нити с помощью опции *num\_threads*, для третьего – четыре нити с помощью вызова функции.

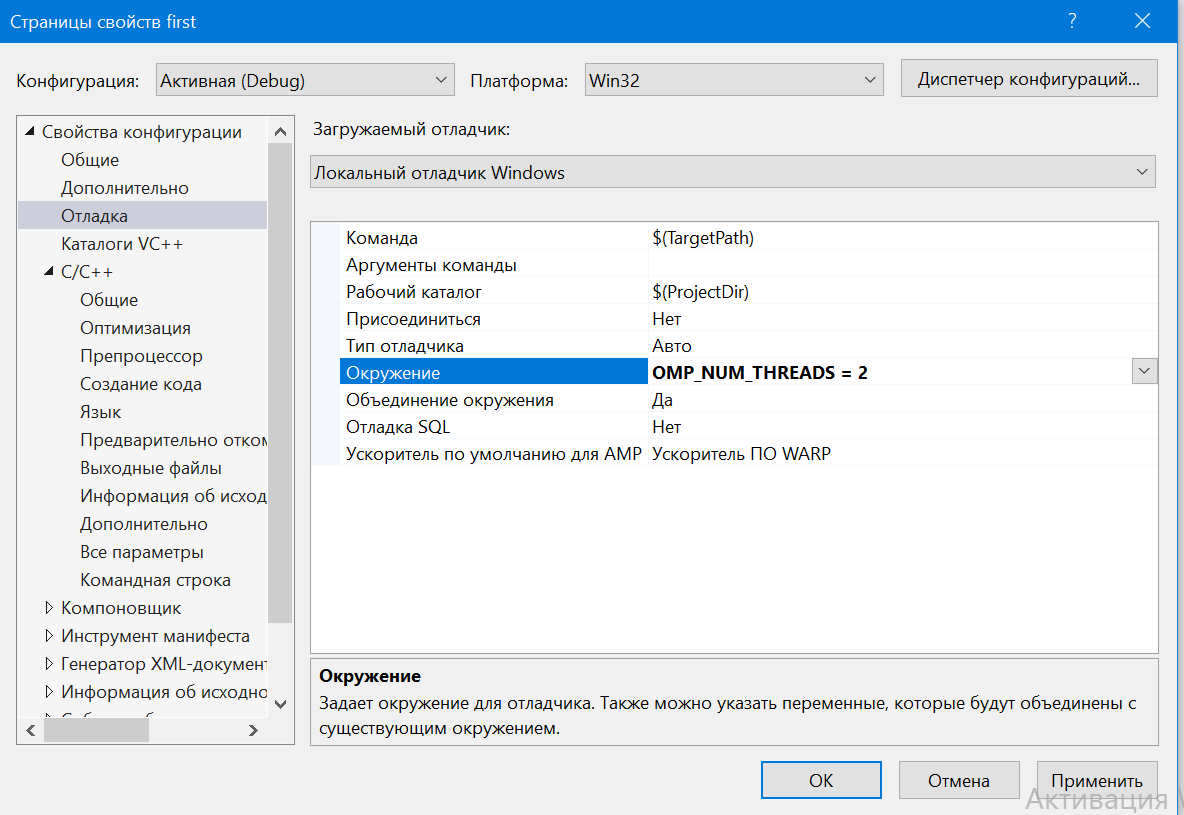


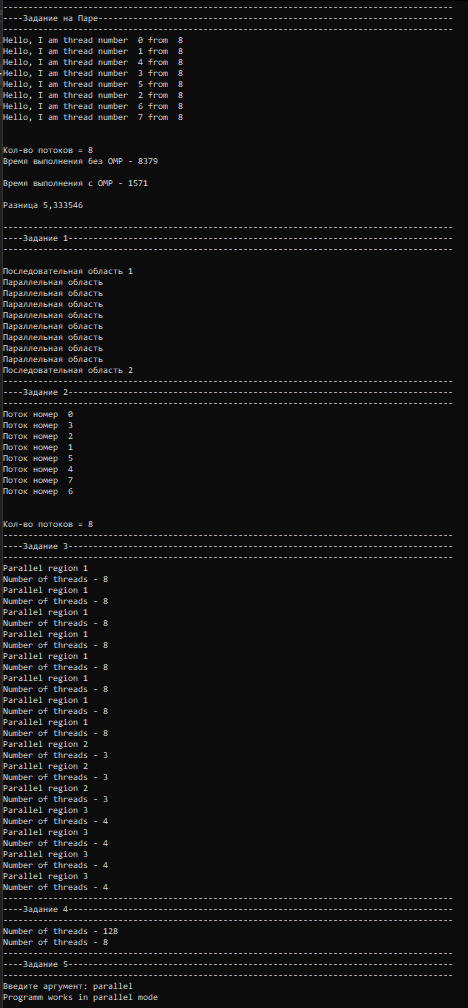
Рисунок 1 установление нитей с помощью пременной окружения

**Задание 5.** Разработать программу, включающую параллельный регион, выполняемый в 128 нитях. Указать вывод количества нитей в потоке-мастере. Проверить количество созданных для исполнения параллельного региона потоков при отключенном и включенном динамическом режиме. Использовать для задания динамического режима переменную среды или вызов функции.

**Задание 6.** Разработать программу, которая принимает из командной строки параметр «parallel» или «serial» и определяет по его значению необходимость запуска параллельного региона или выполнения программы в обычном последовательном режиме. Использовать для реализации программы условную опцию параллельного региона.

Исходный код

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <time.h>  #include <math.h>  #include <string>  using namespace std;  int main()  {  setlocale(LC\_ALL, "Russian");  //------------------------------------------------------------------------------------------  //----Задание на Паре-----------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание на Паре-----------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  int NumOfThreads;    #pragma omp parallel  {  int ThreadId;  ThreadId = omp\_get\_thread\_num();  NumOfThreads = omp\_get\_num\_threads();  printf("Hello, I am thread number % d from % d\n", ThreadId, NumOfThreads);  }  printf("\n\nКол-во потоков = %d\n", NumOfThreads);  long int N = 100000000;  clock\_t t0 = clock();  for (long int i = 0; i < N; i++) {  double j = cos(i);  }  clock\_t t1 = clock() - t0;  printf("Время выполнения без OMP - %d \n\n", t1);  clock\_t t2 = clock();  #pragma omp parallel  {  for (long int i = 0; i < N/8; i++) {  double j = cos(i);  }  }  clock\_t t3 = clock() - t2;  printf("Время выполнения с OMP - %d \n\n", t3);  printf("Разница %lf \n\n", (double)t1/(double)t3);    //------------------------------------------------------------------------------------------  //---------Задание 1------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 1-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  printf("\r\nПоследовательная область 1\r\n");  #pragma omp parallel  {  printf("Параллельная область\r\n");  }  printf("Последовательная область 2\r\n");  //------------------------------------------------------------------------------------------  //---------Задание 2------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 2-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  #pragma omp parallel  {  int ThreadId;  ThreadId = omp\_get\_thread\_num();  NumOfThreads = omp\_get\_num\_threads();  printf("Поток номер % d\n", ThreadId);  }  printf("\n\nКол-во потоков = %d\n", NumOfThreads);  //------------------------------------------------------------------------------------------  //---------Задание 3------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 3-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  #pragma omp parallel  {  printf("Parallel region 1\n");  printf("Number of threads - %d\n", omp\_get\_num\_threads());  }  #pragma omp parallel num\_threads(3)  {  printf("Parallel region 2\n");  printf("Number of threads - %d\n", omp\_get\_num\_threads());  }  omp\_set\_num\_threads(4);  #pragma omp parallel  {  printf("Parallel region 3\n");  printf("Number of threads - %d\n", omp\_get\_num\_threads());  }  //------------------------------------------------------------------------------------------  //---------Задание 4------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 4-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  omp\_set\_dynamic(0);  #pragma omp parallel num\_threads(128)  {  if (omp\_get\_thread\_num() == 0) {  printf("Number of threads - %d\n", omp\_get\_num\_threads());  }  }  omp\_set\_dynamic(1);  #pragma omp parallel num\_threads(128)  {  if (omp\_get\_thread\_num() == 0) {  printf("Number of threads - %d\n", omp\_get\_num\_threads());  }  }  //------------------------------------------------------------------------------------------  //---------Задание 5------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 5-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  printf("Введите аргумент: ");  string argv;  cin >> argv;  if (strcmp(argv.c\_str(), "parallel") == 0) {  printf("Programm works in parallel mode\n");  #pragma omp parallel  {  }  }  if (strcmp(argv.c\_str(), "serial") == 0) {  printf("Programm works in sequential mode\n");  }  //------------------------------------------------------------------------------------------  } |

Пример выполнения программы

Ответы на вопросы

1. Каким образом связаны между собой процессы и потоки в многозадачной операционной системе?

Процессы статичны и состоят из набора потоков, исполняющих программный код, и области данных. Процесс может содержать много потоков, но обязательно содержит, по крайней мере, один. Когда системе предлагают программу «запустить на исполнение» она создает один процесс и один поток, в котором и запускает программу.

1. Каковы преимущества выполнения многопоточных приложений на многопроцессорных вычислительных системах?

Многопоточность позволяет нескольким потокам выполняться в рамках одного процесса, совместно используя ресурсы процесса и при этом работая самостоятельно. Применяя многопоточную модель для разделения одного процесса на несколько потоков, становится возможным организовать его параллельное выполнение на многопроцессорной системе. Это преимущество многопоточной программы позволяет ей работать быстрее на компьютерных системах, которые имеют несколько процессоров или процессор с несколькими ядрами

1. Что такое технология OpenMP?

Технология OpenMP является на сегодняшний день наиболее популярным средством для программирования на масштабируемых SMP-системах (SSMP, ccNUMA и других) в модели общей памяти (*shared memory model*). Особенностью данной технологии является возможность естественного преобразования последовательной программы в параллельную с помощью библиотечных процедур, директив и переменных окружения

1. Опишите класс вычислительных систем, для которых предназначена OpenMP модель программирования.

OpenMP (Open Multi-Processing) — это набор директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью (SMP-системах).

1. Что такое «главный» и «подчиненный» потоки?

OpenMP реализует параллельные вычисления с помощью многопоточности, в которой «главный» (master) поток создает набор «подчиненных» (slave) потоков, и задача распределяется между ними. Предполагается, что потоки выполняются параллельно на машине с несколькими процессорами, причём количество процессоров не обязательно должно быть больше или равно количеству потоков.

1. Каким образом осуществляется подключение OpenMP?

Для включения *OpenMP* нажмите *Project* → *Properties.* Слева вверху появившегося окна выберите «*All Configurations*» и в разделе *Configuration Properties* → *C/C++* → *Language* включите «*OpenMP Support*»

1. Как определить подключение библиотеки OpenMP?

Для проверки того, что компилятор поддерживает какую-либо версию **OpenMP**, достаточно написать директивы условной компиляции *#ifdef* или *#ifndef*. Простейший пример условной компиляции приведен в примере

1. Опишите формат директив компилятору.

Директивы **OpenMP** в программах оформляются указаниями препроцессору, начинающимися с *#pragma omp*. Ключевое слово *omp* используется для того, чтобы исключить случайные совпадения имён директив **OpenMP** с другими именами.

Формат директивы:

**#pragma** omp *имя\_директивы* [опция[[,] опция]...]

1. Опишите различие между последовательными и параллельными регионами?

Программа начинается с *последовательной области* – сначала работает один процесс (нить), при входе в *параллельную область* порождается ещё некоторое число процессов, между которыми в дальнейшем распределяются части кода. По завершении параллельной области все нити, кроме одной (нити-мастера), завершаются, и начинается последовательная область

1. Каково назначение нити-мастера и порожденных нитей? Каково время их существования в системе?

Программа начинается с *последовательной области* – сначала работает один процесс (нить), при входе в *параллельную область* порождается ещё некоторое число процессов, между которыми в дальнейшем распределяются части кода. По завершении параллельной области все нити, кроме одной (нити-мастера), завершаются, и начинается последовательная область

1. Что такое блок, ассоциированный с оператором?

Объектом действия большинства директив является один оператор или блок, перед которым расположена директива в исходном тексте программы. В **OpenMP** такие операторы или блоки называются *ассоциированными* с директивой

1. Каким образом описывается параллельная секция?

Параллельная область задаётся при помощи директивы parallel:

**#pragma** omp parallel[опция[[,]опция]...]

1. Каким образом определить номер потока, выполняющего программный код?

Функция *omp\_get\_thread\_num()* позволяет нити получить свой уникальный номер в текущей параллельной области.

int omp\_get\_thread\_num(void);

1. Опишите способы определения количества потоков для выполнения параллельного региона. Входит ли в это число поток-мастер?

Задать значение переменной среды OMP\_NUM\_THREADS можно несколькими способами: из командной строки, в настройках проекта, вызовом функции.