**Нижегородский государственный университет**
**им. Н.И.Лобачевского**

***Факультет Вычислительной математики и кибернетики***

***Инструменты параллельного программирования для систем с общей памятью***

# Intel Thread Checker. Краткое описание

Сысоев А.В., Мееров И.Б.
Кафедра математического обеспечения ЭВМ

# Содержание

- Назначение Intel Thread Checker
- Возможности Intel Thread Checker
- Принцип сбора информации
- Подготовка программы для анализа
- Создание проекта в Intel Thread Checker
- Сбор и анализ данных
- Пример использования Intel Thread Checker
- Заключение

# Назначение Intel Thread Checker…

- Процесс отладки в общем случае можно разбить на следующие шаги:
  - определение факта наличия ошибки;
  - поиск (локализация) ошибки;
  - выяснение причин ошибки;
  - определение способа устранения ошибки;
  - устранение ошибки.
- ITC помогает в первых четырех случаях.

# Назначение Intel Thread Checker

- Назначение
  - поиск мест с возможным недетерминированным поведением многопоточной программы
- Тип приложений
  - POSIX threads, WinAPI threads, OpenMP
- ОС
  - Windows, Linux

# Возможности Intel Thread Checker…

- ITC обнаруживает ошибки следующих видов:
  - гонки данных (data races),
  - тупики (deadlocks),
  - потоки в состоянии ожидания (stalled threads),
  - потерянные сигналы (lost signals),
  - заброшенные замки (abandoned locks).
- Рассмотрим далее эти ошибки подробнее.

# Возможности Intel Thread Checker. Виды ошибок…

- **Гонки данных**

  Несколько потоков работают с разделяемыми данными и конечный результат зависит от соотношения скоростей потоков.

- **Пример**

  Первый поток выполняет над общей переменной **x** операцию **x = x + 3**. Второй поток – операцию **x = x + 5**.

  Данная операция фактически разбивается на три отдельных подоперации: считать **x** из памяти, увеличить **x**, записать **x** в память.

  В зависимости от взаимного порядка выполнения потоками подопераций финальное значение переменной **x** может быть больше исходного на 3, 5 или 8.

# Возможности Intel Thread Checker. Виды ошибок…

- **Тупики**

  Взаимная блокировка потоков, ожидающих наступление некоторого события для продолжения работы.

  Типичный пример тупика: нулевой поток занял для использования ресурс 1 и ожидает предоставления ему ресурса 2; первый поток занял ресурс 2 и ожидает предоставления ему ресурса 1.

# Возможности Intel Thread Checker. Виды ошибок…

- **Потоки в состоянии ожидания**

  Одно из состояний потока в многозадачной операционной системе – ожидание.

  Поток переходит в него, когда для продолжения выполнения ему требуется наступление некоторого внешнего события.

  Если пребывание потока в этом состоянии продолжается слишком долго, ITC рапортует об ошибке типа stalled thread.

  Интервал времени, по истечении которого выдается данная диагностика, может быть задан в настройках ITC.

# Возможности Intel Thread Checker. Виды ошибок

- **Потерянные сигналы**

  Возникают, когда поток ожидает наступление некоторого события, произошедшего прежде, чем поток пришел в состояние готовности к его приему и обработке. В результате поток никогда не сможет выйти из состояния ожидания.

- **Заброшенные замки**

  Возникают в ситуации, когда поток захватил некоторый ресурс (критическую секцию, мьютекс) и был снят с выполнения по той или иной причине. В результате ресурс не может быть освобожден. Если он требуется другому потоку, это приведет к бесконечному ожиданию.

# Принцип сбора информации

- Инструментация кода
  - вставка обращений к библиотеке записи и анализа исполнения кода

  (!) должна быть выключена оптимизация

- Контролируется
  - доступ к памяти
  - операции синхронизации
  - операции создания потоков

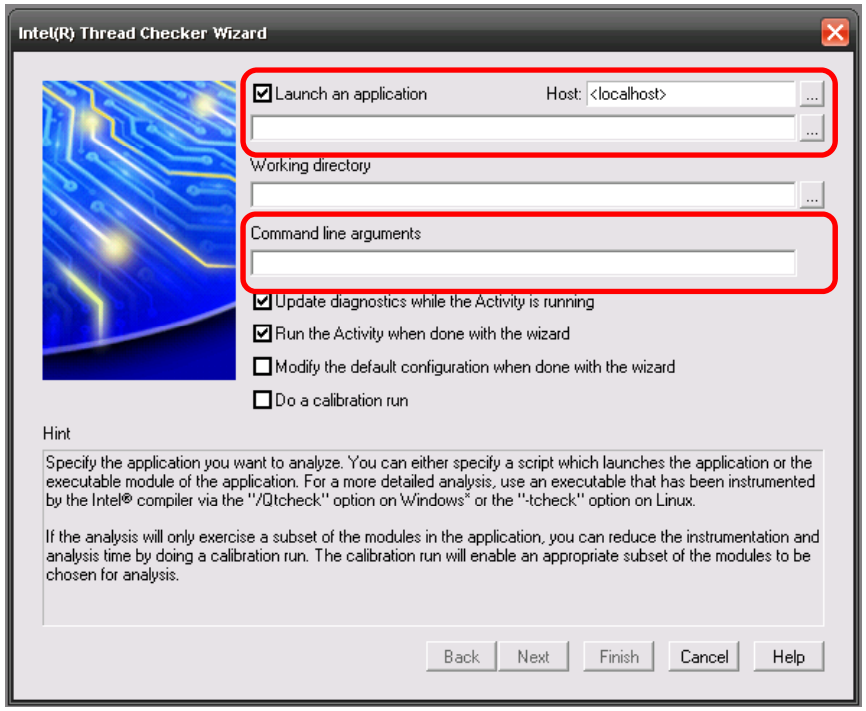
  (!!) неисполняемые участки не проверяются

# Подготовка программы для анализа

- Использование отладчика Intel Thread Checker возможно в двух режимах:
  - бинарная инструментация программы (осуществляется автоматически в момент запуска Activity)
  - компиляторная инструментация программы (необходимо указать ключ компилятора /Qtcheck)
- Сборка приложения для работы с ITC предполагает установку следующих опций проекта (или настроек в make-файле):
  - компиляция потоко-безопасного кода: -MT[d], -MD[d]
  - использование debug опций: -Z[i,I,7] –Od
  - связывание с ключом /fixed:no

# Создание проекта в Intel Thread Checker

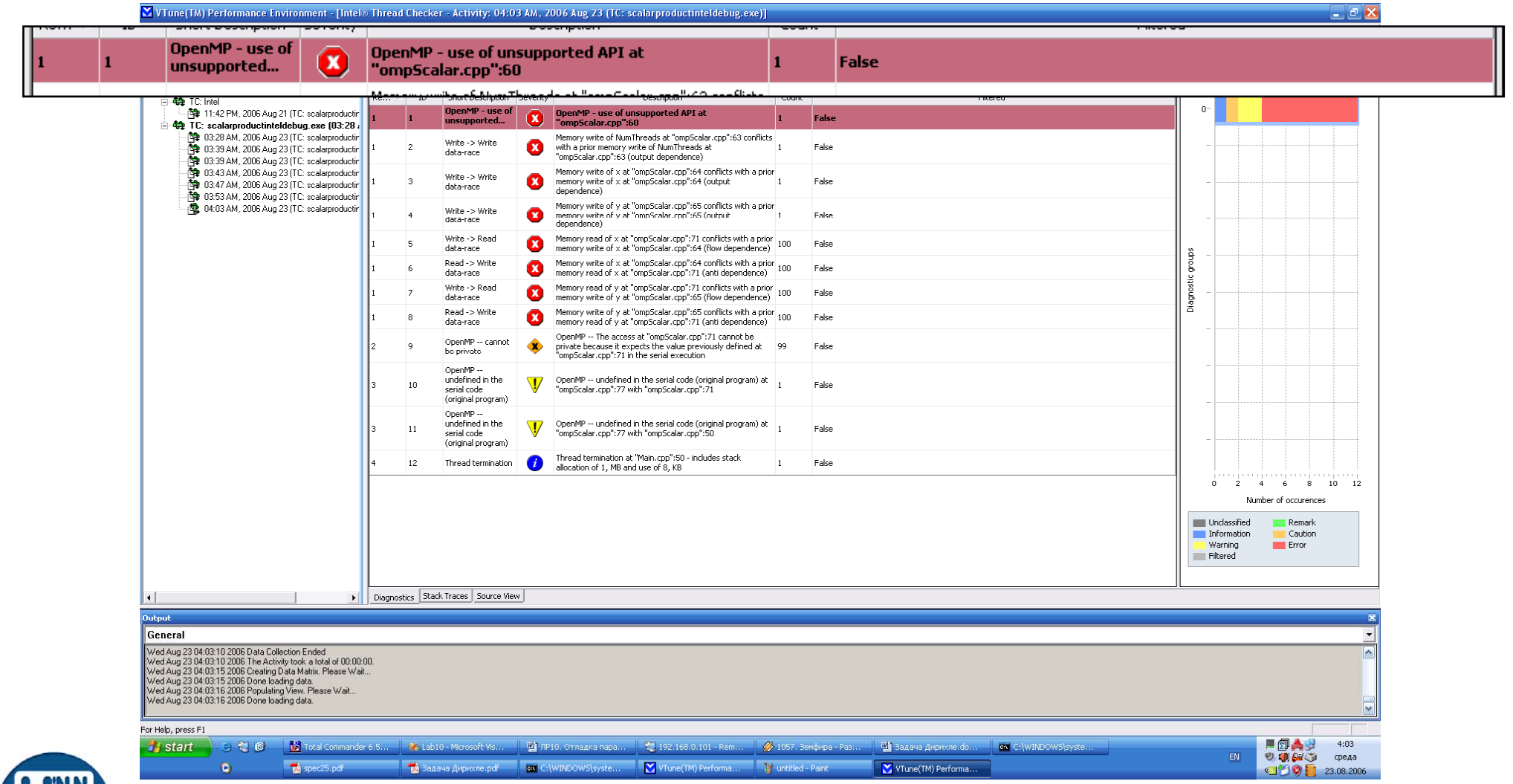
- Команда меню File->New Project
- Указать исполняемый файл (Launch an application)
- При необходимости указать аргументы командной строки




# Сбор и анализ данных…

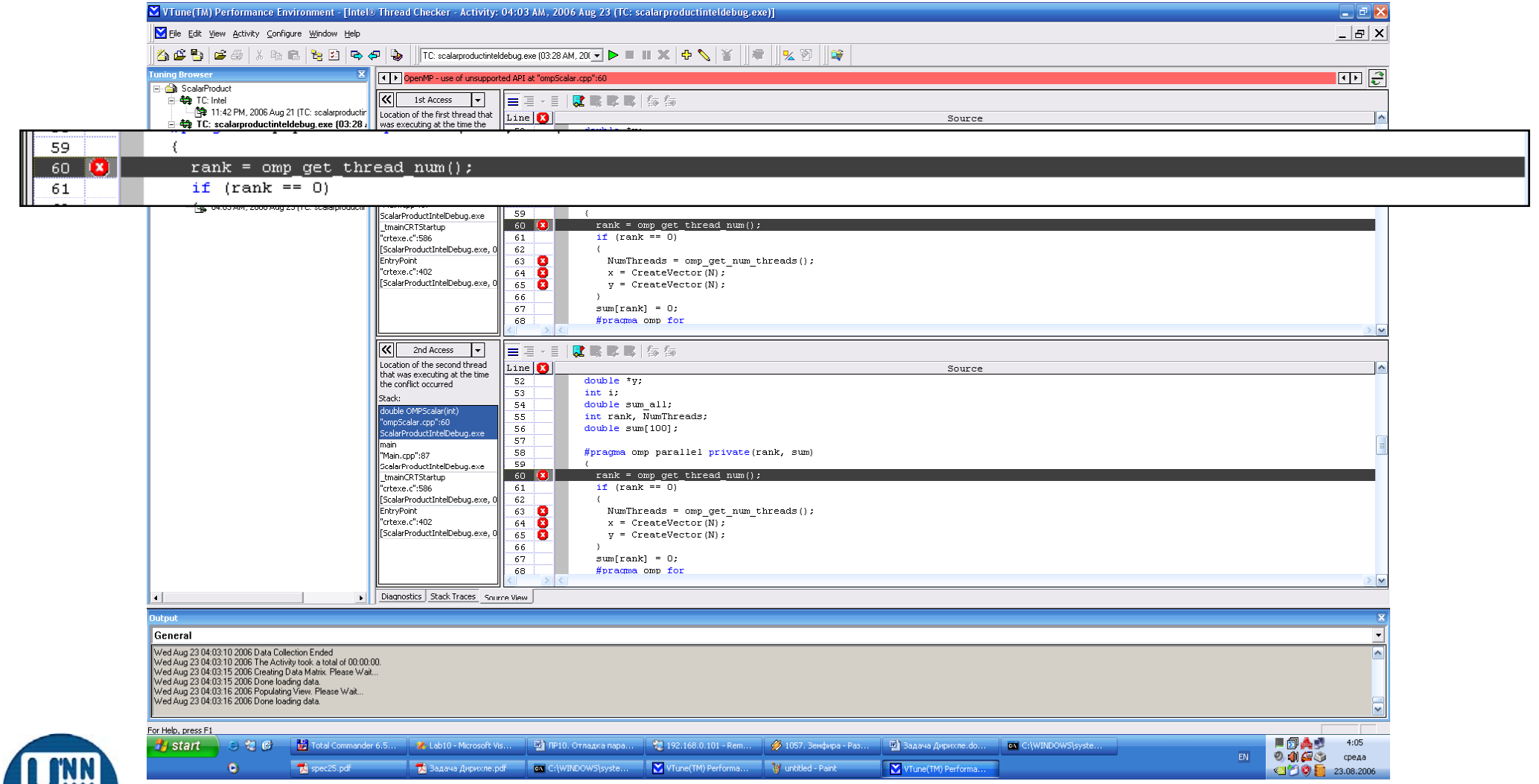
- После запуска в проекте ITC активности начинается инструментация исполняемого модуля и используемых им динамических библиотек.
- Затем модуль запускается и начинается процесс анализа. По завершении ITC формирует окно с информацией о найденных ошибках и подозрительных местах.
- Повторный запуск (один из 3-х вариантов):
  - Выбрать пункт меню **Activity→Run**.
  - Нажать **F5**.
  - Нажать кнопку  на панели инструментов.

# Сбор и анализ данных…



# Сбор и анализ данных

- По каждой диагностике можно получить информацию:



# Пример использования ITC. Описание примера

- Пример из поставки ITC.
- Создает 4 потока, каждый из которых увеличивает значение общей глобальной переменной **`globalX`** и использует критическую секцию для синхронизации доступа к ней.
- Несмотря на наличие критической секции, в коде содержится конфликт доступа (гонки данных). Нам предстоит его обнаружить, выяснить причину и исправить ситуацию.

# Пример использования ITC. Изучение примера…

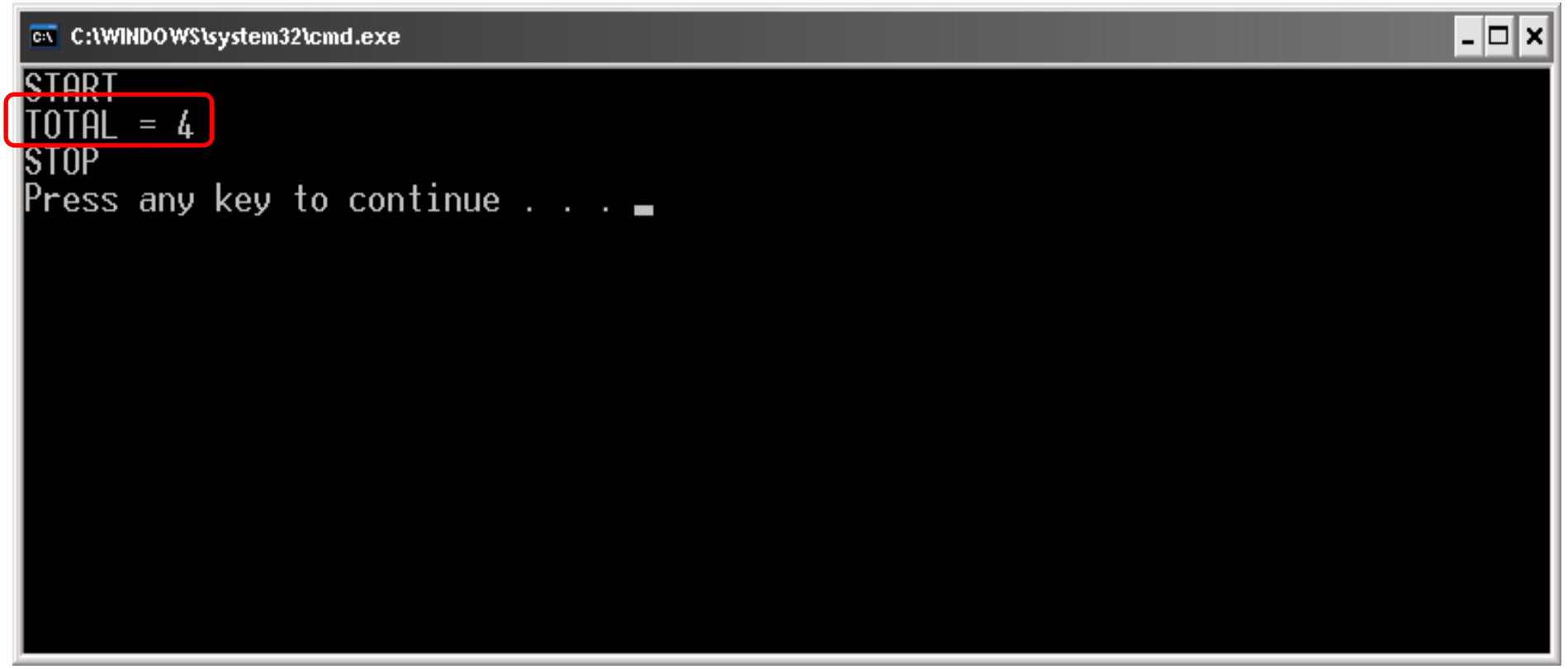
- Файл DataRaces.c

```
int globalX = 0;
DWORD WINAPI increment (void *arg)
{
  CRITICAL_SECTION cs;
  InitializeCriticalSection (&cs);
  EnterCriticalSection (&cs);
    globalX++;
  LeaveCriticalSection (&cs);

  DeleteCriticalSection (&cs);
  return 0;
}
```

# Пример использования ITC. Изучение примера…

- Запустим пример на выполнение.




- Результат выглядит правильным.

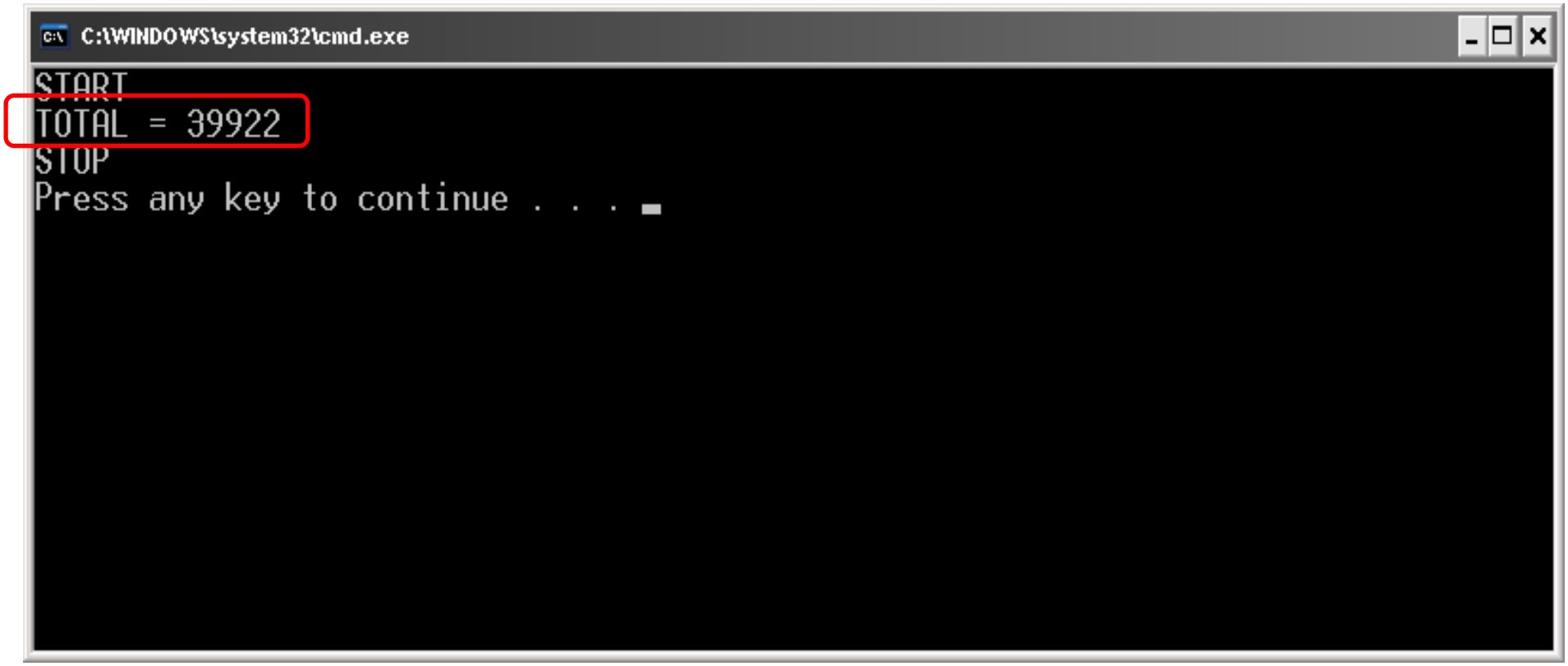
# Пример использования ITC. Изучение примера…

- Изменим немного код. Пусть каждый поток увеличивает значение переменной **globalX** не один раз, а многократно.

```
DWORD WINAPI increment (void *arg)
{
  int i;
  CRITICAL_SECTION cs;
  InitializeCriticalSection (&cs);
  for (i = 0; i < 10000; i++)
  {
    EnterCriticalSection (&cs);
      globalX++;
    LeaveCriticalSection (&cs);
  }
  DeleteCriticalSection (&cs);
  return 0;
}
```

# Пример использования ITC. Изучение примера…

- Результат работы (при разных запусках может отличаться):




**ГОНКИ ДАННЫХ!**

# Пример использования ITC. Подготовка программы для анализа…

1. Верните код примера к исходному состоянию.
2. Конвертируйте проект для использования компилятора Intel® C++ Compiler. В окне **Solution Explorer** выберите файл проекта, щелкните правой кнопкой мыши и в контекстном меню выполните команду **Convert to use Intel® C++ Project System**.
3. В меню **Project** выберите пункт **Properties**, в появившемся окне настроек проекта в дереве слева выберите узел **Configuration Properties→C/C++→General**. В открывшейся таблице справа убедитесь, что значение поля **Debug Information Format** равно **Program Database (/Zi)**.

# Пример использования ITC. Подготовка программы для анализа…

4. В дереве слева выберите узел **Configuration Properties→C/C++→Optimization**. В открывшейся таблице справа убедитесь, что значение поля **Optimization** равно **Disabled(/Od)**.
5. В дереве слева выберите узел **Configuration Properties→C/C++→Code Generation**. В открывшейся таблице справа убедитесь, что значение поля **Runtime library** установлено в **Multi-threaded Debug DLL (/MD)**.

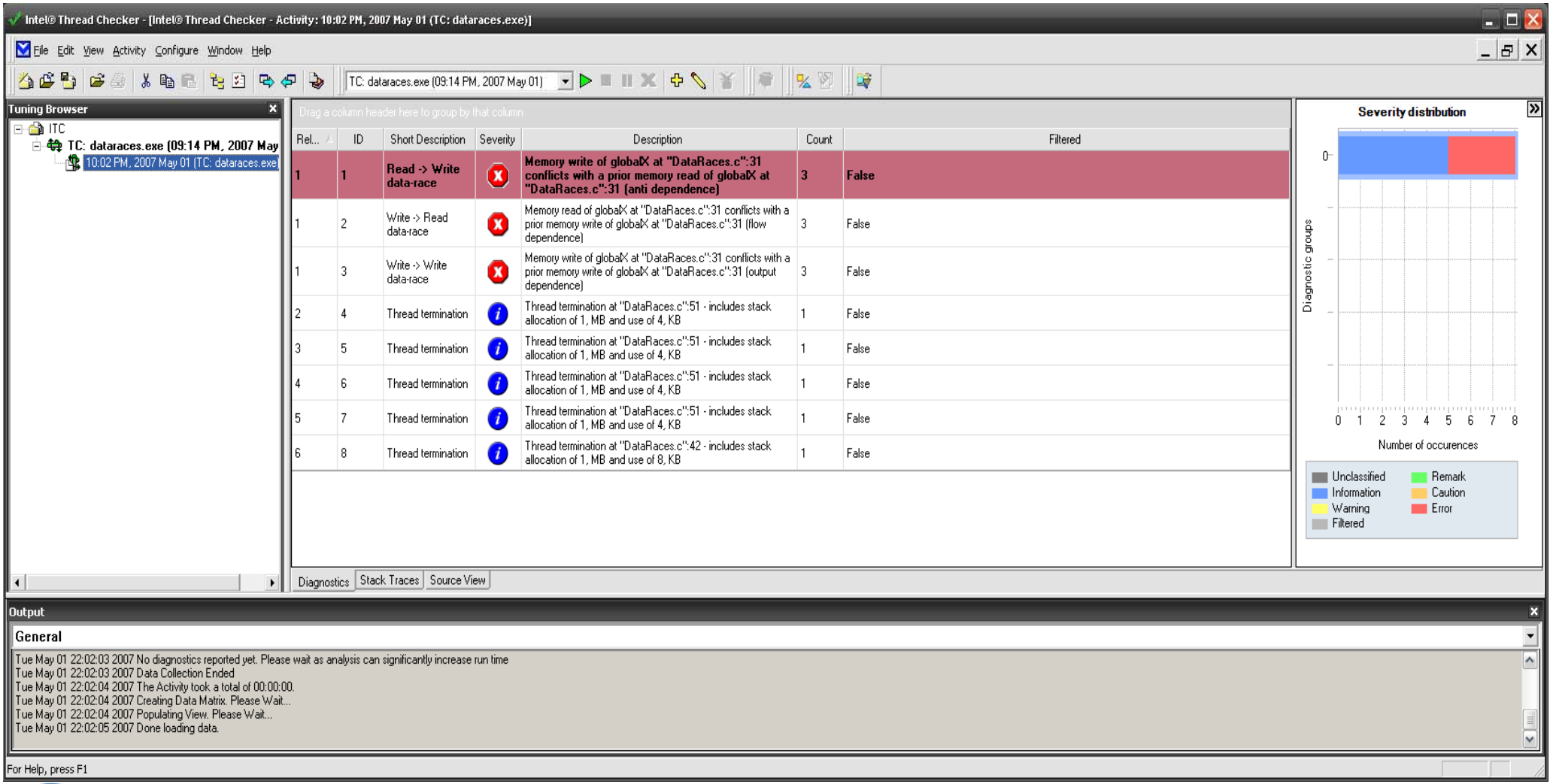
# Пример использования ITC.
# Подготовка программы для анализа

6. В дереве слева выберите узел **Configuration Properties→Linker→Command Line**. Убедитесь, что сборка программы выполняется с использованием опции компоновщика **/FIXED:NO**.
7. Убедитесь, что включена компиляторная инструментация кода. В дереве слева выберите узел **Configuration Properties→C/C++→Command Line**. Убедитесь, что в поле **Additional Options** установлен ключ компилятора **/Qtcheck**.

# Пример использования ITC.
# Создание проекта в ITC

1. Запустите Intel® Thread Checker. Найти его можно, например, по следующему пути: **Start→All programs→Intel(R) Software Development Tools→Intel(R) Thread Checker 3.0→Intel(R) Thread Checker**.
2. В открывшемся окне нажмите на кнопку **New Project** .
3. В окне создания проекта выберите **Intel® Thread Checker Wizard** и нажмите кнопку **OK**.
4. В окне мастера в поле **Launch an application** укажите путь к исполняемому файлу **C:\ITCLabs\DataRaces\Debug\DataRaces.exe**.
5. Нажмите кнопку **Finish**.

# Пример использования ITC. Анализ…

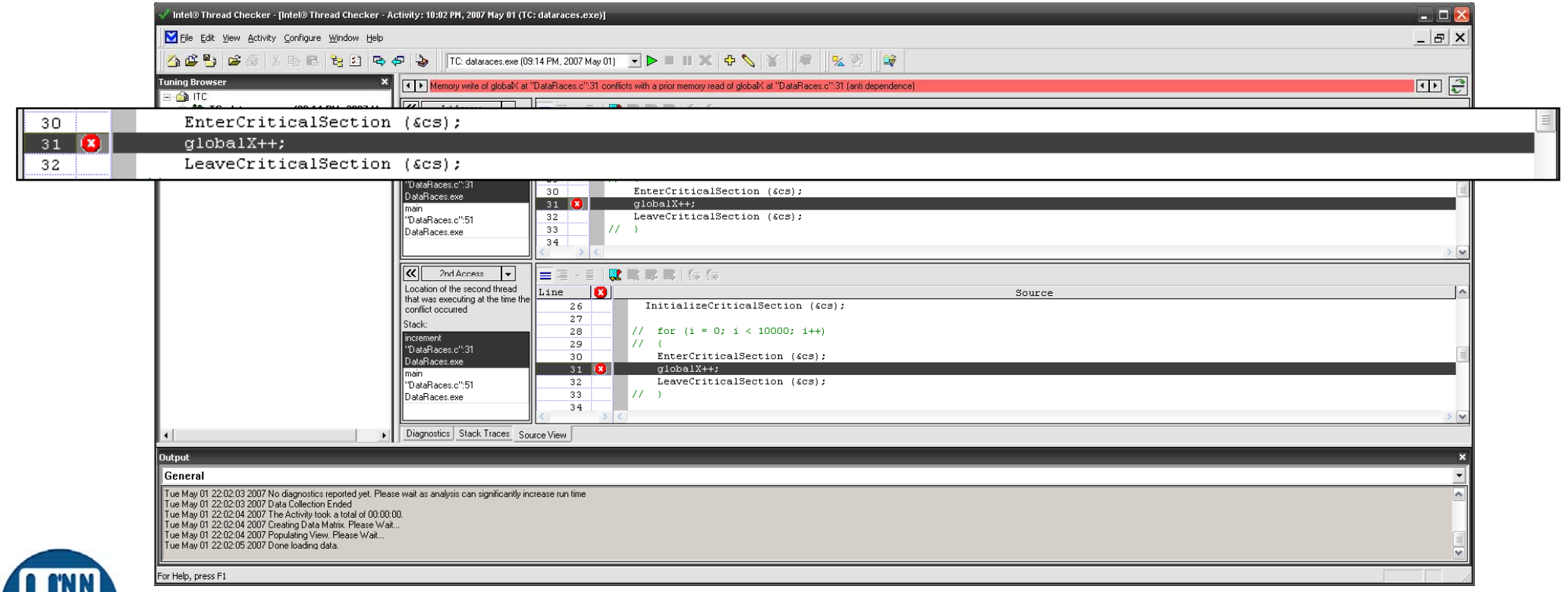


# Пример использования ITC. Анализ…

- ITC нашел в предложенном коде 3 ошибки.
- При ближайшем рассмотрении видно, что все они указывают на одну и ту же переменную **`globalX`**.
- Краткие комментарии к диагностикам показывают, что ITC указал все возможные комбинации неверного обращения к переменной **`globalX`**.
  - Когда первый поток в нее пишет, а второй читает.
  - Когда первый читает, второй пишет.
  - И, наконец, когда оба пишут.
- Несмотря на то, что реально работало 4 потока, очевидно, что для демонстрации ошибки достаточно двух. Именно так ITC всегда и комментирует гонки данных.

# Пример использования ITC. Анализ…

- При наличии отладочной информации ITC может показать в исходном коде местоположение ошибки. Выберите любую из найденных ошибок и двойным щелчком по ней перейдите к просмотру исходного кода.



# Пример использования ITC. Причина и устранение…

**Причина**:

- Противоречие с тем фактом, что обращение потоков к переменной **`globalX`** происходит внутри критической секции, а гонка данных все равно имеется, кажущееся.
- Проблема в данном случае не в критической секции, а в объекте, на котором она основана (**`CRITICAL_SECTION cs`**).
- Этот объект объявлен внутри потоковой функции, а значит, является локальным для каждого потока. Когда один поток захватывает критическую секцию, он делает это для своего локального объекта **`cs`**, к которому остальные потоки не имеют доступа.

# Пример использования ITC. Причина и устранение…

**Устранение**:

- Исправить ситуацию можно несколькими способами, но в любом из них объявление объекта **`cs`** нужно вынести из потоковой функции **`increment`**, а инициализацию критической секции и освобождение ресурсов поместить в функцию **`main`**.

# Пример использования ITC. Причина и устранение

```
int globalX = 0;
CRITICAL_SECTION cs;
DWORD WINAPI increment (void *arg)
{
  EnterCriticalSection (&cs);
    globalX++;
  LeaveCriticalSection (&cs);
  return 0;
}

int main (int argc, char *argv[])
{
  HANDLE h[NTHREADS]; DWORD  rc;
  int i;
  printf ("START\n");
  InitializeCriticalSection (&cs);
  for (i = 0; i < NTHREADS; i++)
  {
    h[i] = CreateThread (0, 0, increment, NULL, 0, NULL);
  }
  rc = WaitForMultipleObjects (NTHREADS, h, TRUE, INFINITE);
  DeleteCriticalSection (&cs);
  printf ("TOTAL = %d\n", globalX); printf ("STOP\n");
}
```

# Заключение

- Ни для кого не секрет, наличие инструментов делает жизнь проще.
- Перефразируем: наличие Intel® Thread Checker существенно скрашивает суровые будни разработчика многопоточных программ.

# Темы заданий для самостоятельной работы

- По приведенному выше сценарию изучите примеры из поставки ITC:
  - Deadlock – пример, иллюстрирующий тупики и, кроме того, содержащий ситуацию гонки данных между главным и дочерними потоками
  - HelloWorld – пример, демонстрирующий гонки данных при передаче информации в потоковые функции

# Литература


- Intel® Thread Checker for Windows*. Getting Started Guide. Version 3.0. — Intel Corporation, 2006.
- Intel® Thread Checker Help. Version 3.0. — Intel Corporation, 2006.
- Intel® Thread Checker. Guide to Sample Code. Version 3.0. — Intel Corporation, 2006.


# Авторский коллектив

- Сысоев Александр Владимирович
- Мееров Иосиф Борисович