



**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.
- ИТС 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...

- ИТС 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, ИТС рапортует об ошибке типа stalled thread.

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



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.

❑ Зброшенные 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



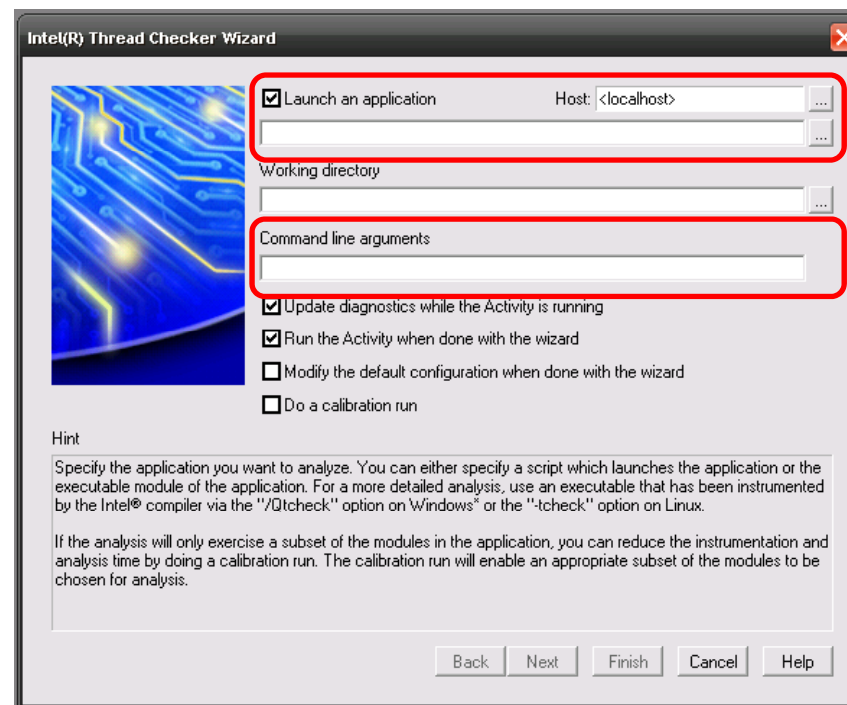
Подготовка программы для анализа

- 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 ИТС предполагает установку следующих опций проекта (или настроек в make-файле):
 - компиляция потоко-безопасного кода: -MT[d], -MD[d]
 - использование debug опций: -Z[i,l,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 ИТС активности начинается инструментация исполняемого модуля и используемых им динамических библиотек.
- Затем модуль запускается и начинается процесс анализа. По завершении ИТС 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...

VTune(TM) Performance Environment - [Intel® Thread Checker - Activity: 04:03 AM, 2006 Aug 23 (TC: scalarproductintdebug.exe)]

Item ID	Count	Severity	Description	Code	Filtered
1	1	Error	OpenMP - use of unsupported API at "ompScalar.cpp":60	1	False
1	2	Error	Write -> Write data-race: Memory write of NumThreads at "ompScalar.cpp":63 conflicts with a prior memory write of NumThreads at "ompScalar.cpp":63 (output dependence)	1	False
1	3	Error	Write -> Write data-race: Memory write of x at "ompScalar.cpp":64 conflicts with a prior memory write of x at "ompScalar.cpp":64 (output dependence)	1	False
1	4	Error	Write -> Write data-race: Memory write of y at "ompScalar.cpp":65 conflicts with a prior memory write of y at "ompScalar.cpp":65 (output dependence)	1	False
1	5	Error	Write -> Read data-race: Memory read of x at "ompScalar.cpp":71 conflicts with a prior memory write of x at "ompScalar.cpp":64 (flow dependence)	100	False
1	6	Error	Read -> Write data-race: Memory write of x at "ompScalar.cpp":64 conflicts with a prior memory read of x at "ompScalar.cpp":71 (anti dependence)	100	False
1	7	Error	Write -> Read data-race: Memory read of y at "ompScalar.cpp":71 conflicts with a prior memory write of y at "ompScalar.cpp":65 (flow dependence)	100	False
1	8	Error	Read -> Write data-race: Memory write of y at "ompScalar.cpp":65 conflicts with a prior memory read of y at "ompScalar.cpp":71 (anti dependence)	100	False
2	9	Warning	OpenMP -- cannot be private: OpenMP -- The access at "ompScalar.cpp":71 cannot be private because it expects the value previously defined at "ompScalar.cpp":71 in the serial execution	99	False
3	10	Warning	OpenMP -- undefined in the serial code (original program): OpenMP -- undefined in the serial code (original program) at "ompScalar.cpp":77 with "ompScalar.cpp":71	1	False
3	11	Warning	OpenMP -- undefined in the serial code (original program): OpenMP -- undefined in the serial code (original program) at "ompScalar.cpp":77 with "ompScalar.cpp":50	1	False
4	12	Information	Thread termination: Thread termination at "Main.cpp":50 - includes stack allocation of 1, MB and use of 8, KB	1	False

Diagnostic groups: 0-12, Number of occurrences: 0-12

Legend: Unclassified, Information, Warning, Filtered, Remark, Caution, Error

Output window: General, Wed Aug 23 04:03:10 2006 Data Collection Ended, Wed Aug 23 04:03:10 2006 The Activity took a total of 00:00:00, Wed Aug 23 04:03:15 2006 Creating Data Matrix. Please Wait..., Wed Aug 23 04:03:15 2006 Done loading data, Wed Aug 23 04:03:16 2006 Populating View. Please Wait..., Wed Aug 23 04:03:16 2006 Done loading data.

Taskbar: start, Total Commander 6.5..., Lab10 - Microsoft Vis..., ЛР10. Отладка пара..., 192.168.0.101 - Rem..., 1057. Зенфира - Раз..., Задача Дирхле. до..., C:\WINDOWS\systeme..., spec25.pdf, Задача Дирхле.pdf, C:\WINDOWS\systeme..., VTune(TM) Performa..., untitled - Paint, VTune(TM) Performa...



Сбор и анализ данных

- По каждой диагностике можно получить информацию:

The screenshot displays the Intel Thread Checker diagnostic interface. At the top, a red bar indicates the error: "OpenMP - use of unsupported API at 'ompScalar.cpp':60". Below this, a table lists the threads involved in the conflict:

Thread ID	Location
59	{
60	rank = omp_get_thread_num();
61	if (rank == 0)

The main window shows the source code for two threads. The top thread (Thread 59) is at line 60, and the bottom thread (Thread 61) is at line 61. Both threads are executing the same code block:

```
rank = omp_get_thread_num();  
if (rank == 0)  
{  
    NumThreads = omp_get_num_threads();  
    x = CreateVector(N);  
    y = CreateVector(N);  
}  
sum[rank] = 0;  
#pragma omp for
```

The "Stack" pane on the left shows the call stack for the threads, including "main" and "OMPScalar(int)". The "Output" pane at the bottom shows the general diagnostic message:

```
Wed Aug 23 04:03:10 2006 Data Collection Ended  
Wed Aug 23 04:03:10 2006 The Activity took a total of 00:00:00.  
Wed Aug 23 04:03:15 2006 Creating Data Matrix. Please Wait...  
Wed Aug 23 04:03:15 2006 Done loading data.  
Wed Aug 23 04:03:16 2006 Populating View. Please Wait...  
Wed Aug 23 04:03:16 2006 Done loading data.
```



Пример использования ИТС. Описание примера

- Пример из поставки ИТС.
- 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 ИТС. Изучение примера...

□ Файл DataRaces.c

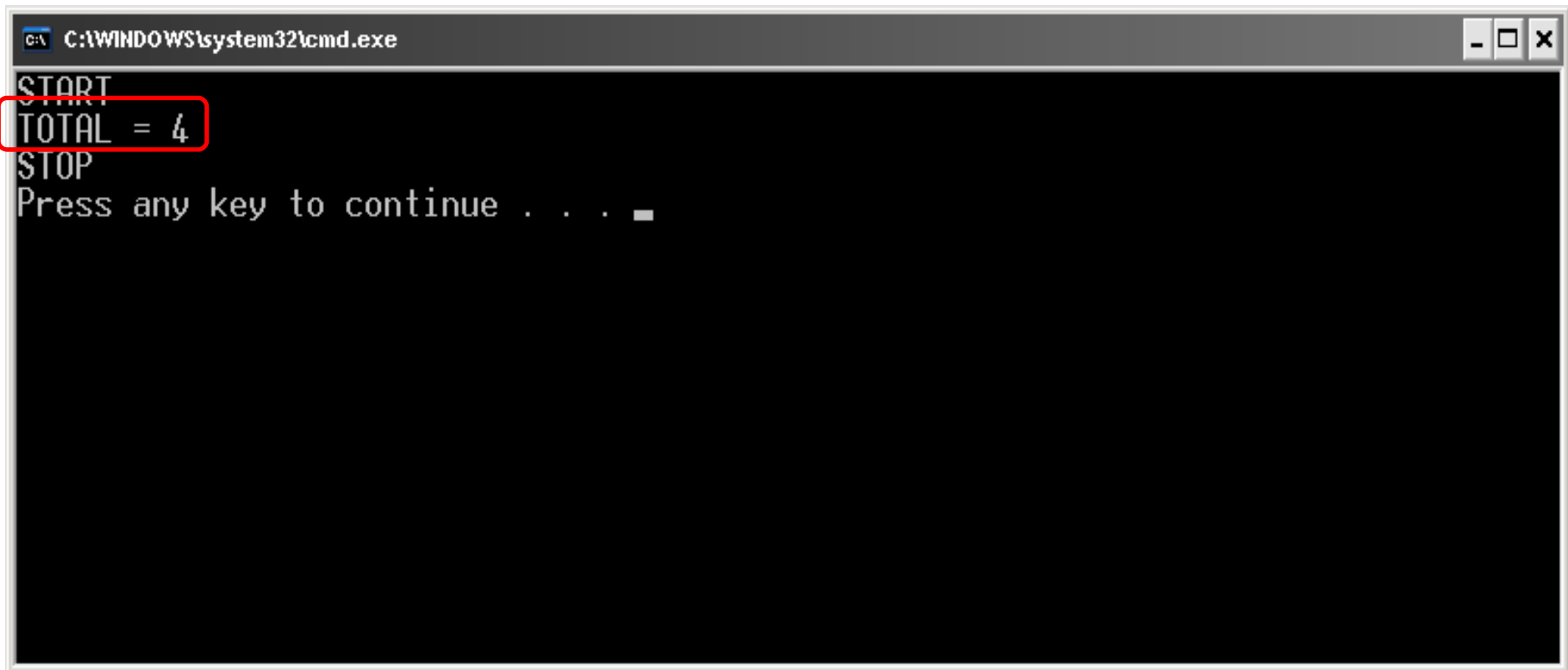
```
int globalX = 0;
DWORD WINAPI increment (void *arg)
{
    CRITICAL_SECTION cs;
    InitializeCriticalSection (&cs);
    EnterCriticalSection (&cs);
    globalX++;
    LeaveCriticalSection (&cs);

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



Пример использования ИТС. Изучение примера...

- ❑ Запустим пример на выполнение.



```
C:\WINDOWS\system32\cmd.exe
START
TOTAL = 4
STOP
Press any key to continue . . .
```

- ❑ Результат выглядит правильным.



Пример использования ИТС. Изучение примера...

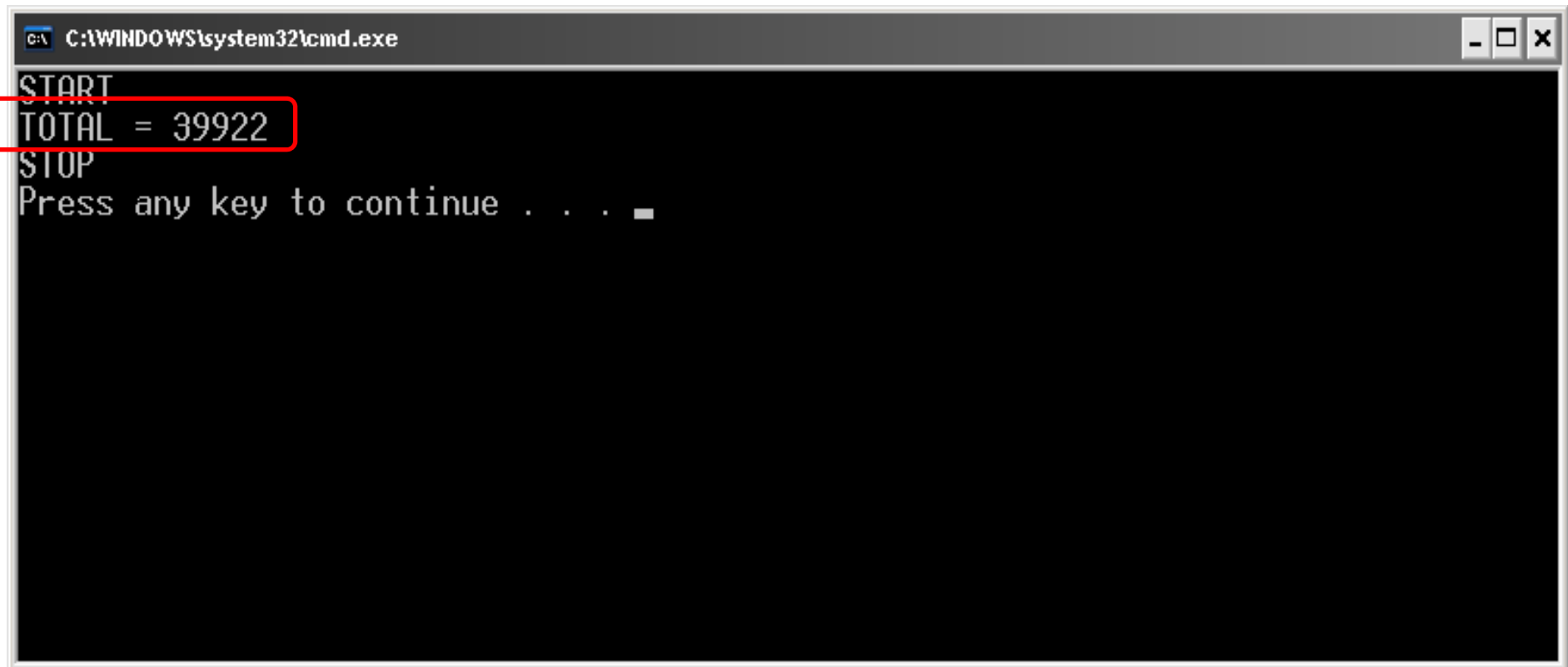
- Изменим немного код. Пусть каждый поток увеличивает значение переменной `globalX` не один раз, а многократно.

```
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 ИТС. Изучение примера...

- ❑ Результат работы (при разных запусках может отличаться):



```
C:\WINDOWS\system32\cmd.exe
START
TOTAL = 39922
STOP
Press any key to continue . . . .
```

ГОНКИ ДАННЫХ!



Пример использования ИТС.

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 ИТС.

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 ИТС.

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 ИТС.

Создание проекта в ИТС

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 ИТС. Анализ...

Intel® Thread Checker - [Intel® Thread Checker - Activity: 10:02 PM, 2007 May 01 (TC: dataraces.exe)]

File Edit View Activity Configure Window Help

TC: dataraces.exe (09:14 PM, 2007 May 01)

Tuning Browser

- ITC
 - TC: dataraces.exe (09:14 PM, 2007 May 01)
 - 10:02 PM, 2007 May 01 (TC: dataraces.exe)

Drag a column header here to group by that column

Rel...	ID	Short Description	Severity	Description	Count	Filtered
1	1	Read -> Write data-race	Error	Memory write of global at "DataRaces.c":31 conflicts with a prior memory read of global at "DataRaces.c":31 (anti dependence)	3	False
1	2	Write -> Read data-race	Error	Memory read of global at "DataRaces.c":31 conflicts with a prior memory write of global at "DataRaces.c":31 (flow dependence)	3	False
1	3	Write -> Write data-race	Error	Memory write of global at "DataRaces.c":31 conflicts with a prior memory write of global at "DataRaces.c":31 (output dependence)	3	False
2	4	Thread termination	Information	Thread termination at "DataRaces.c":51 - includes stack allocation of 1, MB and use of 4, KB	1	False
3	5	Thread termination	Information	Thread termination at "DataRaces.c":51 - includes stack allocation of 1, MB and use of 4, KB	1	False
4	6	Thread termination	Information	Thread termination at "DataRaces.c":51 - includes stack allocation of 1, MB and use of 4, KB	1	False
5	7	Thread termination	Information	Thread termination at "DataRaces.c":51 - includes stack allocation of 1, MB and use of 4, KB	1	False
6	8	Thread termination	Information	Thread termination at "DataRaces.c":42 - includes stack allocation of 1, MB and use of 8, KB	1	False

Severity distribution

Diagnostic groups

Number of occurrences

Legend:

- Unclassified
- Information
- Warning
- Remark
- Caution
- Error
- Filtered

Output

General

```
Tue May 01 22:02:03 2007 No diagnostics reported yet. Please wait as analysis can significantly increase run time
Tue May 01 22:02:03 2007 Data Collection Ended
Tue May 01 22:02:04 2007 The Activity took a total of 00:00:00.
Tue May 01 22:02:04 2007 Creating Data Matrix. Please Wait...
Tue May 01 22:02:04 2007 Populating View. Please Wait...
Tue May 01 22:02:05 2007 Done loading data.
```

For Help, press F1



Пример использования ИТС. Анализ...

- ❑ ИТС 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 ИТС 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 ИТС всегда и комментирует гонки данных.



Пример использования ИТС. Анализ...

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

The screenshot displays the Intel Thread Checker interface. At the top, a red error message reads: "Memory write of globalX at 'DataRaces.c':31 conflicts with a prior memory read of globalX at 'DataRaces.c':31 (anti dependence)". Below this, a code snippet shows lines 30, 31, and 32:

```
30 EnterCriticalSection (&cs);
31 globalX++;
32 LeaveCriticalSection (&cs);
```

The main window shows a source code view with the following code:

```
26 InitializeCriticalSection (&cs);
27
28 // for (i = 0; i < 10000; i++)
29 // {
30 EnterCriticalSection (&cs);
31 globalX++;
32 LeaveCriticalSection (&cs);
33 // }
34
```

The interface also includes a stack trace on the left, a diagnostics pane at the bottom, and an output window showing logs from May 01, 2007.



Пример использования ИТС.

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 ИТС. 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 ИТС.

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 ИТС:
 - 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

