Нижегородский государственный университет
им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Образовательный комплекс

Технологии разработки параллельных программ

Intel Thread Profiler
Лабораторная работа №2
Синхронизация и накладные расходы на поддержку многопоточности

Корняков К.В., Шишков А.В.
Кафедра математического обеспечения ЭВМ

Цель лабораторной работы

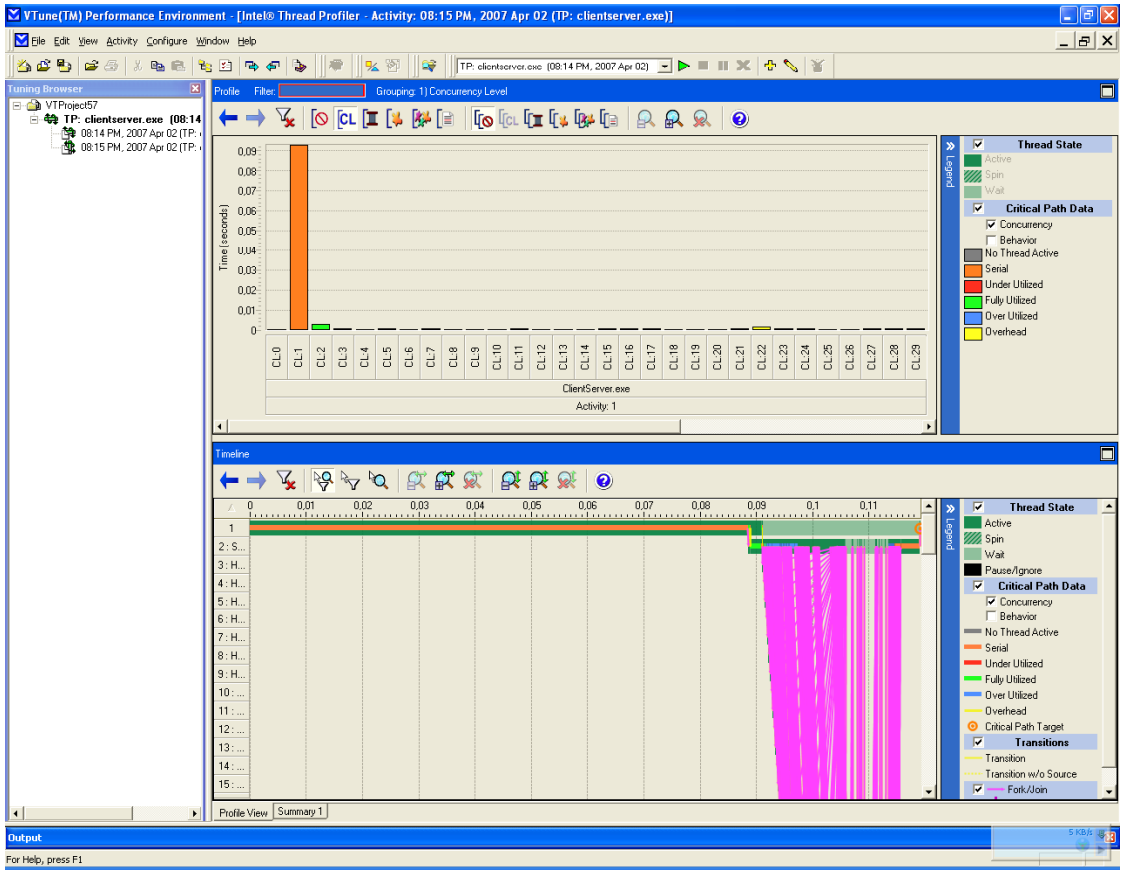
- Изучить основные методы снижения накладных расходов на поддержку многопоточности.
- Изучить вопросы, связанные с выбором и использованием примитивов синхронизации.

Задание 1

- Откройте проект **ClientServer** в **Microsoft Visual Studio 2005**.
- Подготовьте приложение к профилированию.
- Ознакомьтесь с кодом приложения:
  1. **`main`** – рабочая функция потока-клиента;
  2. **`ServerThreadFunc`** – рабочая функция потока-сервера;
  3. **`HandlerPoolThreadFunc`** – рабочая функция потока-обработчика.
- Создайте проект ITP.
- Запустите процесс профилирования.
- Проанализируйте полученные результаты.

Подход 1: обработка каждого запроса в отдельном потоке

Результаты



- Зафиксируйте время работы

Архитектура многопоточного сервера

Варианты обработки запросов

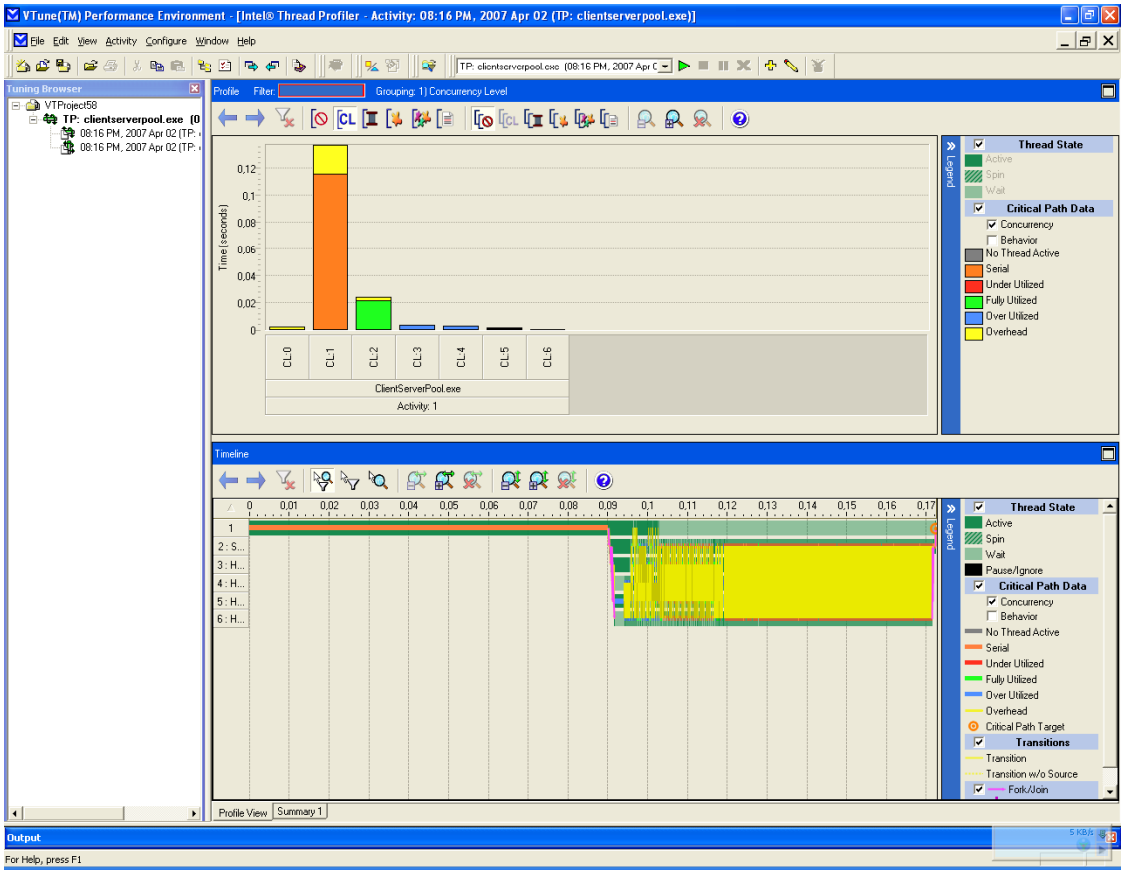
- обработка всех запросов в одном потоке
- обработка каждого нового запроса в отдельном потоке
- организация пула потоков

Задание 2

- Вернитесь **Microsoft Visual Studio**.
- Выберите проект **ClientServerPool**.
- Ознакомьтесь с кодом приложения.
- Вернитесь в ITP и запустите процесс профилирования.
- Проанализируйте полученные результаты.

Подход 2: организация пула потоков

Результаты



- Зафиксируйте время работы

Задание 3

- Вернитесь **Microsoft Visual Studio**.
- Закомментируйте критическую область для доступа к **`requestsStatistics`.**
- Раскомментируйте вызов функции **`InterlockedIncrement`.**
- Вернитесь в ITP и запустите процесс профилирования.
- Проанализируйте полученные результаты.

Снижение частоты синхронизации

В нашем приложении синхронизация происходит слишком часто (большое количество стрелок желтого цвета).

Можно предложить следующее решение:

- вместо одной очереди сообщений создается **`numTypes`** очередей для запросов одинакового типа;
- сервер для каждого запроса, определяет его тип и помещает его в соответствующую очередь;
- каждый поток обрабатывает сообщения одного типа;
- каждый поток работает только со своим счетчиком, поэтому пропадает необходимость в синхронизации доступа к массиву **`requestsStatistics`**.

Самостоятельная работа

- Реализуйте модель синхронизации с использованием критических секций.
- Ответьте на контрольные вопросы
  - В чем преимущество создания пула потоков перед обработкой всех запросов в одном потоке и обработкой каждого запроса в новом потоке?
  - В чем недостатки использования примитивов синхронизации уровня ядра ОС?
  - Как еще можно увеличить эффективность приложения **ClientServerPool**?

Источники информации

1. «Developing Multithreaded Applications: A Platform Consistent Approach», Intel Corporation, March 2003.
2. «Threading Methodology: Principles and Practices», Intel Corporation, March 2003.
3. «Multi-Core Programming for Academia», Student Workbook, by Intel.
4. «Multi-Core Programming», book by Sh. Akhter and J. Roberts, Intel Press 2006.
5. «Intel® Thread Profiler. Getting Started Guide».
6. «Intel® Thread Profiler. Guide to Sample Code».
7. «Intel® Thread Profiler Help».
8. «Intel® Thread Profiler – краткое описание», материалы по образовательному комплексу «Технологии разработки параллельных программ».
9. «Эффективная многопоточность», Алексей Ширшов, RSDN Magazine #2-2003.